Titre: Title:	XCS Algorithms for a Linear Combination of Discounted and Undiscounted Reward Markovian Decision Processes
Auteur: Author:	Maryam Moghimi
Date:	2018
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Moghimi, M. (2018). XCS Algorithms for a Linear Combination of Discounted and Undiscounted Reward Markovian Decision Processes [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/3694/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/3694/
Directeurs de recherche: Advisors:	Samuel Jean Bassetto, & Jean-Marc Frayret
Programme: Program:	Maîtrise recherche en génie industriel

UNIVERSITÉ DE MONTRÉAL

XCS ALGORITHMS FOR A LINEAR COMBINATION OF DISCOUNTED AND UNDISCOUNTED REWARD MARKOVIAN DECISION PROCESSES

MARYAM MOGHIMI

DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INDUSTRIEL)

AOÛT 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

XCS ALGORITHMS FOR A LINEAR COMBINATION OF DISCOUNTED AND UNDISCOUNTED REWARD MARKOVIAN DECISION PROCESSES

présenté par : MOGHIMI Maryam

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. ADJENGUE Luc, Ph. D., président

M. BASSETTO Samuel, Doctorat, membre et directeur de recherche

M. FRAYERT Jean-Marc, Ph. D., membre et codirecteur de recherche

M. PARTOVI NIA Vahid, Doctorat, membre

DEDICATION

To whom I love…

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Prof. Samuel Bassetto and my co-supervisor Prof. Jean-Marc Frayret for their support, patience, motivation and help to complete this project successfully.

I also would like to express my sincere thanks to my thesis committee: Professor Vahid Partovi nia and Professor Luc Adjengue for kindly evaluating my thesis.

Finally, I wish to thank my parents, my brothers, and my friends at CIMAR LAB for their support and their kindness.

RÉSUMÉ

Plusieurs études ont montré que combiner certains prédicteurs ensemble peut améliorer la justesse de la prédiction dans certains domaines comme la psychologie, les statistiques ou les sciences du management. Toutefois, aucune de ces études n'ont testé la combinaison de techniques d'apprentissage par renforcement. Notre étude vise à développer un algorithme basé sur deux algorithmes qui sont des formes approximatives d'apprentissage par renforcement répétés dans XCS. Cet algorithme, MIXCS, est une combinaison des techniques de Q-learning et de R-learning pour calculer la combinaison linéaire du payoff résultant des actions de l'agent, et aussi la correspondance entre la prédiction au niveau du système et la valeur réelle des actions de l'agent. MIXCS fait une prévision du payoff espéré pour chacune des actions disponibles pour l'agent.

Nous avons testé MIXCS dans deux environnements à deux dimensions, Environment1 et Environment2, qui reproduisent les actions possibles dans un marché financier (acheter, vendre, ne rien faire) pour évaluer les performances d'un agent qui veut obtenir un profit espéré. Nous avons calculé le payoff optimal moyen dans nos deux environnements et avons comparé avec les résultats obtenus par MIXCS.

Nous avons obtenu deux résultats. En premier, les résultats de MIXCS sont semblables au payoff optimal moyen pour Environments1, mais pas pour Environment2. Deuxièmement, l'agent obtient le payoff optimal moyen quand il prend l'action "vendre" dans les deux environnements.

ABSTRACT

Many studies have shown that combining individual predictors improved the accuracy of predictions in different domains such as psychology, statistics and management sciences. However, these studies have not tested the combination of reinforcement learning techniques. This study aims to develop an algorithm based on two iterative approximate forms of reinforcement learning algorithm in XCS. This algorithm, named MIXCS, is a combination of Q-learning and R-learning techniques to compute the linear combination payoff and the correspondence between the system prediction and the action value. As such, MIXCS predicts the payoff to be expected for each possible action.

We test MIXCS in two two-dimensional grids called Environment1 and Environment2, which represent financial markets actions of buying, selling and holding to evaluate the performance of an agent as a trader to gain the desired profit. We calculate the optimum average payoff to predict the value of the next movement in both Environment1 and Environment2 and compare the results with those obtained with MIXCS.

The results show that the performance of MIXCS is close to optimum average reward in Environment1, but not in Environment2. Also, the agent reaches the maximum reward by taking selling actions in both Environments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AXCS	XCS with a method of average reward
CASs	Complex adaptive systems
GA	Genetic algorithm
LCSs	Learning classifier systems
MDF	Markov decision process
MIXCS	Linear combination of XCSG and XCSAR
RL	Reinforcement leaning
XCS	Extended classifier systems
XCSAR	XCS with a method of average reward
XCSG	XCS with gradient
A	Action space
$R: S \times A \rightarrow \mathbb{R}$	Reward function
S	State space
$T: S \times A \rightarrow S$	Transition probability function
t	Time step
$\pi: S \rightarrow A$	Policy

CHAPTER 1 INTRODUCTION

1.1 Motivation

Our world is based on the interplay of different elements and factors which therefore cannot be regarded individually but rather as a whole; these systems are defined as “complex system” and have the capacity of changing and learning from experiences. A common way to describe such systems are rules, and they can represent these systems; since rules are virtually defined as an accepted means of expressing knowledge for decision making (Holmes, Lanzi, Stolzmann, & Wilson, 2002). The question arises which single best-fit model most suitable in dealing with these complex systems. One possible and powerful solution is so-called rule-based agents, agent is a single component of a given system to interact with the world as an environment of the problem domain. An intelligent agent model that interacts with the environment and improves adaptively with experience is called learning classifier system. Learning is the source of improvement and promotes the system to provide payoff from the environment (Urbanowicz & Moore, 2009).

Artificial markets are a rising form of agent-based social simulation in which agents represent individual consumers, traders, firms or industries interacting under simulated market conditions. Agent-based social simulation is particularly applicable for studying macroscopic structures like organizations and markets that are based on the distributed interactions of microscopic agents. The structure of the artificial markets is designed according to agent specification, and one way to do this is the ad hoc approach. This approach is often used to explore market behaviour at a high level of abstraction. Like models that are considered for artificially intelligent agent-based social simulation that include an agent or agents which interact with environment and learn and adapt over period, here the environment is a two-dimensional grid in which the agent navigates to find the source of payoff, and where the rules govern the interaction between agents and the environment (Zenobia, Weber, & Daim, 2009).

Payoffs are considered as positive rewards, and rule-based agents aim to maximize the achieved environmental payoffs (Tesfatsion, 2000). While these simple, general models have been useful for demonstrating how complex behavioural patterns can appear from simple elemental mechanisms, their low accuracy has limited their applicability to real markets (Zenobia et al., 2009).

The artificial market needs more advanced innovation forecasting tools to study organizational phenomena (Zenobia et al., 2009) or add more parameters to calculate the accuracy of prediction (S. W. Wilson, 1995). Single models or combinations of the models are applied to help the agent to learn, adapt and take action in the environment.

"In combining the results of these two methods, one can obtain a result whose probability law of error will be more rapidly decreasing." (Laplace, 1818)

This quote shows that combining estimates is not new. Laplace considered combining regression coefficient estimates, one being least squares and the other a kind of order statistic, many years ago. In his work, he compared the properties of two estimators and derived a combining formula. However, he concluded that not knowing the error distribution made this combination inaccessible. The topic has remained of great interest in the community, and considerable literature has accumulated over the years regarding the combination of forecasts (Clemen, 1989).

Hashem and Schmeiser propose another contribution regarding multiple models regarding using optimal linear combinations of some trained neural networks instead of using a single best network. Their results suggest that model accuracy can be improved by combining the trained networks.

The vast available literature about combining models in order to obtain a certain output in very different domains motivates us to also use this approach for combining models respect to the learning classifier systems.

This leads to two main question; how to predict expected payoff of such a combined approach? Furthermore how to accurately relate the input from the environment with a corresponding action for the payoff prediction?

The solution is provided by accuracy-based fitness. XCS is a classifier where each classifier provides an expected payoff prediction. The fitness is calculated based on an inverse function of the classifier's average prediction error. The prediction error is an average of a measure of the error in the prediction parameters. The prediction itself is an average of the payoff received that is updated by a Q-learning-like quantity and the reinforcement learning technique which discounts the future payoff received (Wilson, 1995). What method could replace Q-learning-like quantity to provide a prediction value that has a direct effect on the fitness?

The answer is R-learning. R-learning is similar to Q-learning in form; both are based on iteratively approximating the action values which represent the average adjusted reward of doing an action for the input received (Zang, Li, Wang, & Xia, 2013).

To the best of our knowledge, none of the classifier systems uses a combination of iterative approximation from the table of all action values. This study considers the simple combination of two reinforcement learning techniques to calculate the prediction of the systems and compare its result with a single reinforcement learning technique. This combination is applied in a two-dimensional grid representing an artificial trade market to see the profitable action that agent takes through navigating.

1.2 A brief overview of the proposed methodology

To meet the stated goal, we present the simple average of two techniques Q-learning and R-learning based on ensemble averaging to forecast the prediction of the next movement in two two-dimensional grids. They are assumed as environments, Environment1 and Environment2, where the agent should navigate to reach the considered goal. Figure 1.1 outlines this approach schematically.

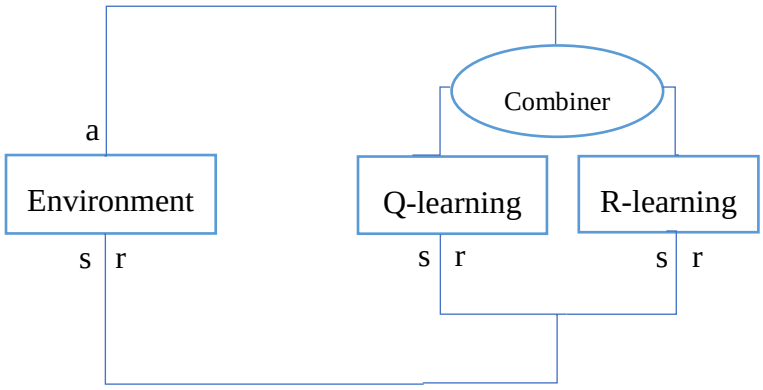

Figure 1-1: Combination of Q-learning and R-learning

Although both environments contain similar objects to represent the rules to learn through exploring and taking action to maximize the profit, they differ regarding size and the actual type of rule objects. The possible action set is divided into two parts, buy and sell, based on the location input received from the agent. The number of buying and selling for similar inputs that maximize profits is calculated. The optimum average of profit is calculated and compared against the single

model approach. The average profit in the combination techniques is close to the optimum average in Environment1, and the results show that maximum profit will be reached by increasing the selling action. Also, a comparison between two single techniques is made to identify the maximum prediction. Since the considered rules in Environment1 are more straightforward than in Environment2, the agent is able to acquire enough knowledge through exploring to take the required actions, whereas the complexity of Environment2 is too high to have a similar effect. As a result, the average profit in this environment diverges greatly from the optimum value for both, the single and the combined approach.

1.3 Thesis structure

In the following chapter 2, sorrow literature is presented. chapter 3 describes the XCS algorithm developed in the course of this thesis. The performance of the proposed algorithm applied to Environment1 and Environment2 is given in chapter 4. Finally, the conclusion and outlook on the future works are presented in chapter 5.

CHAPTER 2 ARTIFICIAL MARKET, REINFORCEMENT LEARNING AND CLASSIFIER SYSTEMS

In this chapter, the artificial market, Markov decision process and the concept of reinforcement learning, reinforcement learning in Markov decision environment, learning classifier systems and the genetic algorithm will be discussed.

2.1 Artificial market

Computer simulation has a long history in simulating organizations by the agent-based paradigm. Agent-based social simulation (ABSS) can fairly well study macroscopic structures like organizations and markets that are based on distributed interactions of microscopic agents. Artificial markets (AMs) are rising a form of ABSS where agents represent individual consumers, firms, trader or industries interacting under simulated market conditions. One of the identified promising applications of AMs in technological innovation is synthesizing and filtering useful information from massive data sets. In this application, agents represent variables and parameters of a data set. These agents will improve through positive feedback or vanish otherwise. The agents also interact and create a new generation of agents that show a higher-order interaction term. As will be shown in the following, an AM could be used to naturally select a population of variables and interaction terms that predict future market behaviour (Zenobia et al., 2009).

The underlying concept is inspired from animal life to find the basic and simple common behaviour between humans and animals when interacting with their environment. Some of these mutualities are for example adaptive behaviour for foraging, navigation and obstacle avoidance in an unknown environment as has been first discussed by Wilson in 1986 (S. W. Wilson, 1986). He offered a bottom-up approach to construct the Animat model from its primitive elements according to four basic characteristics of animals. First, only current sensory signals from the environment are essential for the animal at each moment. Second, the animal is capable of taking action to change these environmental signals. Third, certain signals have a special meaning for the animal, and fourth, the goal of the animal is to externally or internally optimize the rate of occurrence of certain signals (S. W. Wilson, 1986). Needs and satisfaction identify the agent in this model according to these four characteristics.

Based on characteristics, the agent in the Animat model is inspired by real-life rules that are shared between all types of animals, i.e. the manner of interaction with the environment. Based on the first and second characteristics they sense and connect to the environment to satisfy their need. The environment is a simulated world which contains the fundamental objects to serve the surviving task. The animal's ecosystem inspires it. In ABSS, the surviving task of the agents is defined based on the problem, for example, acquiring maximum resources, maintaining a minimum level of energy, prey hunting, obstacle avoidance, etc.

A finite- state machine is one formal way to characterize the environment formally. Two equations define the behaviour of a finite-state machine

$$Q(t+1) = F\big(Q(t), A(t)\big) \qquad (2\text{-}1)$$

$$E(t+1) = G\big(Q(t), A(t)\big) \qquad (2\text{-}2)$$

Where A is the environment input, E is the environment output, and Q represent the current state, time t is assumed to be discrete. The variable A and E are general vectors. The first equation says that the environment's next state is a transition function F of its current state and its defined action. The second equation says that the simulated input at $Q(t)$ for action at t. In general, the model says that the action in an environment results in a new simulated input.

The class of environment is defined depending on the state transition of an agent. If the current state determines the desired action in a state, the agent is placed in a Markovian environment. Otherwise, it needs to have a history of states, and the environment is non-Markovian (S. W. Wilson, 1991). In the next section, a Markovian environment will be discussed in detail.

Agents involved in such a system that consists of a network of interacting agents and exhibit an aggregate behaviour that rises from the individual activities of the agents. It is possible to describe its aggregated behaviour without any knowledge of the individual behaviour of the agent. An agent in such an environment is adaptive if it reaches the goal that is defined by the system. Computer-based adaptive algorithms such as classifier systems, genetic algorithm and reinforcement learning are applied to explore artificial adaptive agents (Holland & Miller, 1991).

In the artificial market, it is necessary to argue how to design the input and output of the agent (Nakada & Takadama, 2013). In this study, an agent is considered as a trader in a stock market where is depicted by two grid environments, one of this environment has a simple 5×5 grid cell

and the other environment 9×9 grid cell that every object is randomly placed. While the agent is randomly located in the environment, its interaction with the environment is analyzed. The task of the agent is to maximize the profit that is provided as Food (F). The action of the agent as a trader is to sell or to buy. There are some objects that are called obstacles (O), they represent the action of hold, the agent cannot trade in the vironment, but in the number of action is considered, and it should choose another action. The possible trade opportunities are demonstrated by $(.)$.The possible action set for this agent is $A = \{1,2,3,4,5,6,7,8\}$, figure 2-1 shows that how the agent $*$ can move around the environment to perform its task.

8 1 2
7 $*$ 3
6 5 4

Figure 2-1: The possible actions in each state for the agent $*$

Each number shows one cell of the environment; it means that agent $*$ can have information of the eight cells at each time step; this information is agent's knowledge that after randomly exploring to learn as much as possible about its environment. After learning the environment, the agent tries to maximize environmental reward in our case profit. In chapter 3, environments are illustrated, and the design of input and output will be explained in details.

2.2 Reinforcement learning

2.2.1 Markov decision processes (MDPs)

A deterministic MDP is defined by its state space S, its action space A, its transition probability function $T: S \times A \to S$, which describes how the state changes as a result of the actions, and its reward function $R: S \times A \to \mathbb{R}$ which evaluates the quality of state transitions. The agent behaves according to the policy $\pi: S \to A$. As a result of the action a_t applied in the state s_t at the discrete time step t the state changes to s_{t+1} according to the transition function $s_{t+1} = T(s_t, a_t)$. At the same time, the agent receives the scalar reward r_{t+1}, according to the reward function $r_{t+1} = R(s_t, a_t)$ where $\|R\|_\infty = sup_{s,a}|R(s,a)|$ is finite. The reward evaluates the immediate effect of the action a_t, namely the transition from s_t to s_{t+1}, but in general does not say anything about its long-term effects (M. L. Puterman, 1994).

The agent chooses actions according to its policy π, using $a_t = \pi(s_t)$. Given T and R, the current state s_t and the current action a_t are sufficient to determine both the next state s_{t+1} and the reward r_{t+1}. To show the Markov property, which is essential in providing theoretical guarantees about reinforcement learning algorithms, suppose X is a random variable that can take its value X_t at time t among the state space $S = \{S_1, S_2, \dots, S_n\}$. The random variable X_t is a Markov chain if:

$$Pr(X_{t+1} = s_{t+1} | X_t = s_t, X_{t-1} = s_{t-1}, \dots, X_1 = s_1, X_0 = s_0) = Pr(X_{t+1} = s_{t+1} | X_t = s_t) \quad (2\text{-}3)$$

It shows the probability distribution of the state at time t+1 depends only on the state at time t (Hohendorff, 2005).

A Markov chain can be shown based on transition probability. Suppose $p(i,j)$ is the probability of going from s_i to s_j by one step, so

$$p(i,j) = \Pr\left(X_{t+1} = s_j \middle| X_t = s_i\right) \qquad (2\text{-}4)$$

The transition probability for all state is considered as

$$\mu(t) = [\mu_1(t)\ \mu_2(t)\ \dots] = [\Pr(X_t = s_1)\ \Pr(X_t = s_2)\ \dots] \qquad (2\text{-}5)$$

The dimension of $\mu(t)$ is as same as the dimension of S. All of the elements of $\mu(0)$ are zero except that one the random variable is in the state. From Chapman-Klmogrov equation,

$$\mu_i(t+1) = \Pr(X_{t+1} = s_i) = \sum_k \Pr(X_{t+1} = s_i | X_t = s_k) \Pr(X_t = s_k)$$

$$= \textstyle\sum_k p(k,i)\mu_k(t) \qquad (2\text{-}6)$$

P is the probability transition matrix that the sum of the rows elements of P is one. So, $\mu(t+1) = \mu(t)P$.

A n-step transition probability $p_{ij}^{(n)}$ is the probability of starting from state i to state j after n states.

It means

$$p_{ij}^{(n)} = \Pr\left(X_{t+n} = s_j \middle| X_t = s_i\right) \qquad (2\text{-}7)$$

Where $p_{ij}^{(n)}$ is the ij^{th} element of P^n.

A vector $\mu = (\mu_1, \dots, \mu_k)^T$ is considered as a stationary distribution for the Markov chain $(X_0, X_1, \dots)$ if

$$\mu \geq 0 \; \forall \; i \in \{1, \dots, k\} \; and \; \sum_{i=1}^{k} \mu_i = 1$$

$$\mu^T P = \mu^T \qquad (2\text{-}8)$$

The current state and next states are independent of initial condition (Hohendorff, 2005).

The reinforcement learning literature often uses “trials” to refer to trajectories starting from some initial state and ending in a terminal state that once reached, can no longer be left.

2.2.2 Definition and basic architecture of reinforcement learning

Dynamic programming (DP) and reinforcement learning (RL) are two algorithmic methods that are applied to solve problems in which actions are used to a system over a period, the time variable is usually discrete, and actions are taken at every discrete time step, to receive the desired goal. DPs methods need to assume the model is known whereas RL methods only require to have access to a set of samples. DPs know the transition probabilities and the expected immediate reward function. Both of these models are useful to obtain behaviour for intelligent agents. If the model of the system cannot be obtained, RLs methods are helpful; since they work using only data obtained from the system without requiring a model of its behaviour (Busoniu, Lucian, et al., 2010).

Reinforcement learning is the problem faced by an agent that must learn how to behave through trial-and-error interactions with an environment. The goal of reinforcement learning is to maximize the rewards and minimize the punishments the agents receive. From the psychological point, the idea of reinforcement learning is inspired by doing the action for a belated reward by animals and human (Richard S. Sutton, 2017).

The main elements of reinforcement learning problems are states (situations), actions that each action influences the agent’s future state and payoffs (rewards or punishments in reinforcements). The agent can take action among the set of possible actions based on a given state, then the agent may transition to a new state and may receive payoffs.

Figure 2-2 presents the architecture of reinforcement learning:

Figure 2-2: Block diagram of the reinforcement learning problem

In this diagram, first, the agent observes the current state of the environment and then chooses an action among the possible action set. In the next step, it receives the immediate reward for its action. To receive the goal of maximization of reward, the agent should try to learn in several times. The agent finds the rule of choosing an action at each state of the environment; this rule is called policy.

More formally, an agent moves in an environment which is characterized by a set S of state, and for each set of state $s \in S$ there is a set of possible actions $A(s_t)$ at a discrete time step t=0, 1, 2…. According to the observed state, the agent chooses an action $a_t \in A(s_t)$. In the next step, t+1 the agent will receive a reward $r_{t+1} \in R$ in the state of s_{t+1}. The agent's goal is to maximize long- term reward, which is defined as the discounted sum of future rewards:

$$\sum_{t=\tau}^{\infty} \gamma^{t-\tau} r_t \qquad 0 \leq \gamma < 1 \qquad (2\text{-}8)$$

Coefficient γ is the discount factor which determines the importance of later and sooner reward. The agent chooses the action according to a policy, the policies are shown by π. If the policy is nondeterministic, giving more actions for a same state, is presented by the probabilistic mapping that is shown, $\pi_t(s,a)$ that represents $a_t = a$ if $s_t = s$. If the policy is deterministic, there is a single action for each state and it is shown by π(s). For each state, an optimal policy gives the best action to perform in that state. When the agent found an optimal policy, it must follow that policy to behave optimally (Maia, 2009).

Since an action effects on the immediate reward, the value of the next state is beneficial to know. In other words, determining the values of states can help in solving the problem. The state- value function, $V^{\pi}(s)$ is the expected value of the discounted sum of future reward when the agent starts from s and follows the policy π.

$$V^{\pi}(s) = E\{\sum_{t=\tau}^{\infty} \gamma^{t-\tau} r_t | s_\tau = s\} \qquad (2\text{-}9)$$

The total expected reward when the agent is in the state $s_t = s$, perform action a, and transition to state s' is divided to two parts, immediate reward $[R(s,a,s')]$ and discounted reward when the state s' starts $\gamma[\gamma V^{\pi}(s')]$. The average of the sum of these presented rewards give the value of the state. In mathematical form:

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s'} T(s,a,s').[R(s,a,s') + \gamma V^{\pi}(s')] \qquad (2\text{-}10)$$

The equation (2-10) is known as the Bellman equation (Maia, 2009).

2.2.3 Reinforcement learning in Markov environment

The problem of reinforcement learning is formalized by using the ideas from dynamical systems theory as the optimal control of Markov decision processes. In dynamic programming (DP) as same as the reinforcement learning, an agent interacts with the environment by the received state, which describes the state of the environment, an action among possible action set, and a scalar reward which provides the feedback on immediate performance. DP and RL problems can be formalized with the help of MDPs (Busoniu, Lucian, et al., 2010).

MDPs straightly indicates the frame of the problem of learning from interaction to achieve a goal. MDPs are a mathematical form of reinforcement learning problem for which precise theoretical statement can be made.

The agent and environment interact at each time steps, $t = 0,1,2,3,\ldots$. The agent receives input from the environment's state $s_t \in S$, based on selects an action $a_t \in A$. One time step later, the agent receive a numerical reward $r_{t+1} \in R$, as the consequence of its action, then the agent will be in a new state s_{t+1}. This is the trajectory of MDP. To show the mathematical expression of a reinforcement problem in Markov transition probability and the expected value of the next reward:

$$P(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a) \qquad (2\text{-}11)$$

$$R(s,a,s') = E(r_{t+1}|s_t = s, a_t = a, s_{t+1} = s') \qquad (2\text{-}12)$$

Transition probability $P(s'|s,a)$ is the probability of state changes from s to s' given action a. The expected value of the next reward $R(s,a,s')$ is the average of receiving a reward in changing from s to s' given action a. The definition of $P(s'|s,a)$ and $R(s,a,s')$ specify the dynamic of a MDP with a finite number of states and actions that is called finite MDP.

The other main element of a reinforcement learning problem except agent and environment is a policy. A policy defines the learning behaviour of an agent at a given time. A policy $\pi: S \rightarrow A$ is a mapping from perceived states of the environment to action set at a given time, it is shown by $\pi_t(s, a)$. To compute the optimal policies for a MDP, dynamic programming is used. The goal of both DP and RL is to find an optimal policy that maximize the return from any initial state s_0. The return is a cumulative aggregation of rewards along a trajectory starting at s_0. The finite-horizon discounted return is given by $\sum_{t=0}^{T} \gamma^t r_{t+1} = \sum_{t=0}^{T} \gamma^t R(s_t, \pi(s_t))$ .It represents the reward obtained by the agent in the long run. Based on the way of accumulating the rewards, there are several types of return.

A convenient way to characterize policies is to define their value functions. There are two types of value function: state-action value functions and state value functions.

1. State-action value function

The state-action value function $V^{\pi}: S \times A \rightarrow \mathbb{R}$ of a policy π gives the return obtained when starting from a given state, applying a given action, and following π thereafter:

$$V^{\pi}(s, a) = E^{\pi}(R_t | s_t = s) = E^{\pi}[(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s)] \qquad (2\text{-}13)$$

Where $(s_0, a_0) = (s, a), s_{t+1} = T(s_t, a_t)\ and\ a_t = \pi(s_t)$. Then, the first term is separated and the rest of the equation is written in a recursive from

$$V^{\pi}(s, a) = R(s, a) + \gamma R^{\pi}(s, a, s') \qquad (2\text{-}14)$$

The optimal value function is defined as the best value function that can be obtained by any policy:

$$V^{*}(s, a) = \max_{\pi} V^{\pi}(s, a) \qquad (2\text{-}15)$$

Any policy π^{*} that selects at each state an action with largest optimal value, i.e., that satisfies:

$$\pi(s) \in argmax v^{*}(s, a) \qquad (2\text{-}16)$$

In general, for a given value function that satisfies (2-16) condition is said to be greedy.

V^{π} and V^{*} are recursively characterized by Bellman equations, which are of central importance for value iteration and policy iteration algorithms. The Bellman equation for V^{π} states that the value of taking action a in state s under the policy π equals the sum of the immediate reward and the discounted value achieved by π in the next state:

$$V^{\pi}(s,a) = R(s,a) + \gamma R^{\pi}(T(s,a), \pi(T(s,a)) \qquad (2\text{-}17)$$

The Bellman optimality equation characterizes V^*, and states that the optimal value of action a taken in state s equals the sum of the immediate reward and the discounted optimal value obtained by the best action in the next state:

$$V^*(s,a) = \mathrm{R(s,a)} + \gamma \max_{a'} V^*(T(s,a), a') \qquad (2\text{-}18)$$

2. State value function :

The state value function $V^{\pi}: S \to \mathbb{R}$ of a policy π is the return obtained by starting a particular state and following π. State value function can be computed from $V^{\pi}(s) = V^{\pi}(s, \pi(s))$. The optimal state value function is the best state value function that can be obtained by any policy, and can be computed from the optimal $V^*(s)$:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad (2\text{-}19)$$

An optimal policy π^* can be computed from V^* by suing the fact that it satisfies:

$$\pi^*(s) \in \operatorname*{argmax}_{a}[R(s,a) + \gamma v^*(T(s,a))] \qquad (2\text{-}20)$$

The V^{π} and V^* satisfy the Bellman equations (Busoniu, Lucian, et al, 2010),

$$V^{\pi}(s) = R(s, \pi(a)) + \gamma V^{\pi}(T(s, \pi(s))) \qquad (2\text{-}21)$$

$$V^*(s) = \max_{a}[\mathrm{R(s,a)} + \gamma V^*\big(T(s,a)\big)] \qquad (2\text{-}22)$$

2.2.4 Temporal differences

Since the agent does not know the MDP, temporal difference learning estimates $V^{\pi}(s)$ by taking the observed value of $R(s,a,s') + \gamma V^{\pi}(s')$ as a sample. In a statistical point of view, the agent likely selects action a after many times to visit state s with probability $\pi(s,a)$. In the same way, the agent chooses a state s' with the transition of $T(s,a,s')$. So, there is an estimate of $V^{\pi}(s)$ that is shown by $\hat{V}^{\pi}(s)$, therefor $R(s,a,s') + \gamma \hat{V}^{\pi}(s')$.

It is essential to consider the potential changes in recent samples to estimate $V^{\pi}(s)$. The simple solution is to weight the recent samples by exponential recency- weighted average iteratively. Let

$x_1, x_2, \dots, x_n$ be a sequence of number and $\bar{x}_n$ is an exponential recency- weighted average of these numbers, $\bar{x}_{n+1} = \bar{x}_n + \alpha[x_{n+1} - \bar{x}_n]$ is an exponential recency-weighted average of $x_1, x_2, \dots, x_n, x_{n+1}$ (Maia, 2009). So, it is possible to update $\hat{V}^\pi(s)$,

$$\hat{V}^\pi(s) \leftarrow \hat{V}^\pi(s) + \alpha[R(s,a,s') + \gamma\hat{V}^\pi(s') - \hat{V}^\pi(s)] \quad (2\text{-}23)$$

The prediction error is defined by the difference between the sample of the discounted sum of future reward and one that is predicted. It is presented as following

$$\delta = R(s,a,s') + \gamma\hat{V}^\pi(s') - \hat{V}^\pi(s) \quad (2\text{-}24)$$

A form of (2-23) based on prediction error is:

$$\hat{V}^\pi(s) \leftarrow \hat{V}^\pi(s) + \alpha\delta \quad (2\text{-}25)$$

$$\hat{V}^\pi(s) \leftarrow (1-\alpha)\hat{V}^\pi(s) + \alpha[R(s,a,s') + \gamma\hat{V}^\pi(s')] \quad (2\text{-}26)$$

Also, this form indicates the new estimate of $\hat{V}^\pi(s)$ is a weighted average of the old estimate and the provided estimate by the current sample.

2.2.5 Q-Learning

The Bellman optimality equation is applied by value iteration techniques to iteratively compute an optimal value function, form which an optimal policy is derived. Q-learning, the most wildly used algorithm from model-free value iteration algorithm. Q-learning begins from an arbitrary initial value and updates it without requiring a model. Instead of a model, Q-learning uses state transitions and rewards, a data tuples $s_t, a_t, s_{t+1}, r_{t+1}$. This is the simple way to find a policy and value function is to store action value $Q(s,a)$ for each state s and action a (Watkins & Dayan, 1992). After each transition, the Q-function is updated by using a data tuple, as follows:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)] \quad (2\text{-}27)$$

where α is a learning factor, a small positive number. The term between square brackets is the temporal difference, the difference between the updated estimate $r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a')$ of the optimal Q-value of (s_t,a_t) and the current estimate $Q_t(s_t,a_t)$. The other form of this updating is:

$$Q_{t+1}(s_t,a_t) = (1-\alpha)Q_t(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{a} Q_t(s_{t+1},a)] \quad (2\text{-}28)$$

The value of Q is unchanged for all other combination of s and a. This method is a form of value iteration, one of the normal dynamic programming algorithms. According to Watkin and Dayan, this method will converge rapidly to the optimal action- value function for finite Markov decision process. He proofs Q-learning converges with probability one under an artificial controlled Markov process called the action replay process (ARP) which is constructed from the trajectory sequence and the learning rate sequence (Watkins & Dayan, 1992).

2.2.6 R-Learning

Before presenting R-learning, it is crucial to introduce the average reward optimality criteria. As it is mentioned in Q-learning section, receiving rewards in the future are geometrically discounted by the discount factor. The average reward model is an undiscounted reinforcement learning that is supposed to take actions to maximize long-run average reward per time step.

$$\rho = \rho(s) = \lim_{N\to\infty} \frac{E(\sum_{t=0}^{N-1} r_t(s))}{N} \qquad (2\text{-}29)$$

Since it is supposed to calculate ρ for the N period, the upper bound of the summation is considered based on $N-1$ previous steps. We expect that the long-run average reward ρ is gained by

$$\lim_{N\to\infty} E\left(\sum_{t=0}^{N-1}\left(r_t(s)\right)\right) \qquad (2\text{-}30)$$

To show the long-run time for future reward $N \to \infty$ is considered.

R-learning is a similar technique to Q-learning in form. R-learning is based on iteratively approximating all action values, the more common points of these methods will be mentioned. R-learning represents the average adjusted reward of doing an action a in state s and related policy to reach the maximize reward. Following rule shows the update R value:

$$R_{t+1}(s_t, a_t) = (1-\beta_R)R_t(s_t, a_t) + \beta_R[r_{t+1} + \max_a R_t(s_{t+1}, a) - \rho] \qquad (2\text{-}31)$$

Here, it should update the average reward ρ according to the following rule:

$$\rho = \left(1-\beta_\rho\right)\rho + \beta_\rho(r_{t+1} + \max_{a\in A} R_{t+1}(s_{t+1}, a) - \max_{a\in A} R_t(s_t, a)) \qquad (2\text{-}32)$$

here, $0 \le \beta_R \le 1$ is the learning rate for updating action values R (.,.), and $0 \le \beta_\rho \le 1$ is the learning rate for updating reward ρ (Zang et al., 2013).

2.3 Learning classifier systems

2.3.1 Introduction

Our world and many systems consist of interconnected parts so that some properties are not defined by the properties of individaul parts. These “complex systems” show a large number of interacting components, whose collective activity is nonlinear. One of the features of these systems is their ability to become adaptive; meaning that they can change and learn from experiences. Therefore, these systems are called complex adaptive systems (CASs). Rule-based agents represent them; the term agent is used to refer to a single component of a given system generally. In general, CASs might be seen as a group of interacting agents where a collection of simple rules can represent each agent's behaviour. These rules typically are demonstrated by “If condition Then action.” Rules generally use information from the system’s environment to make decisions.

2.3.2 What is a learning classifier system and how does it work?

Knowledge of the problem domain describes the learning classifier systems (LCSs) algorithm; this algorithm is seeking a single best-fit model when dealing with the complex systems. The LCSs outputs are classifiers to model an intelligent decision maker collectively. In general, a LCS is an intelligent agent model that interacts with an environment and improves adaptively with experience. The algorithm improves by reinforcement due to payoff by the environment. A LCS aims to maximize the achieved environmental payoffs.

Adaptivity and generalization are two characters that support learning classifier systems. Because of changing situations, LCS has always been viewed as adaptive systems. In recent years, there is some evidence in a different domain such as computational economics to prove this characteristic (Tesfatsion, 2000). In LCSs, generalization is achieved through the evolution of general rules; it means that a classifier can match more than one input vector of the environment. Different situations are maybe recognized with similar consequences. Scalability is the other characteristic of learning time and system size in a different environment that there is no clear answer (Holmes et al., 2002).

The four components of LCSs consist of

1) a finite population of condition-action- rules that called classifiers that represent the current knowledge of the system,

2) a performance component, which regulates the interaction between environment and classifier population,

3) a reinforcement component or credit assignment component which distributes the reward received from the environment to the classifiers and 4) a discovery component which uses different operators to discover better rules and improve existing one. These component represent an algorithmic framework.

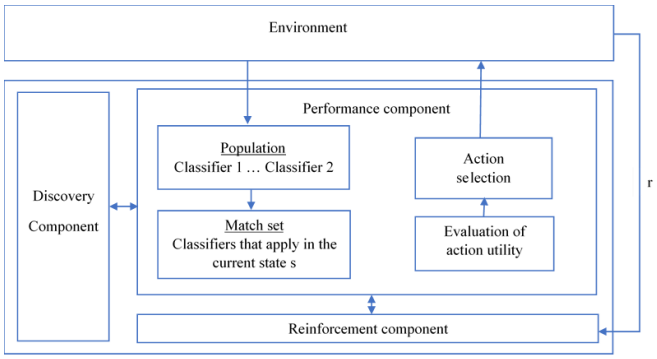

Figure 2-3: The learning classifier systems and environment

2.3.3 The discovery mechanisms and learning mechanisms

The four components of LCSs represent an algorithmic framework, and these mechanisms are responsible for driving the system. For driving the system, discovery and learning are two responsible mechanisms. Discovery mechanism refers to the rule that does not exist in the population, and each rule offers a better rule to get the payoff. Learning is a process to build a general model through experiences by interacting with the environment.

2.3.3.1 Discovery

As it mentioned, the discovery mechanism refers to rule discovery that does not exist in the population, and each rule offers a better rule to get the payoff. Making a good decision have been achieved by a genetic algorithm (GA). The GA is a computational search technique, which manipulates a population of rules to represent a potential solution to a given problem. GA is applied to classifier systems to evolve rules and create new rules that are called evolution. LCSs can be

used to solve reinforcement learning problems, classification problems, and function approximation problems (Urbanowicz & Moore, 2009).

Two measures, the prediction and fitness, are associated with classifiers. Prediction estimates the classifier utility if the classifier is used and fitness estimates the quality of the information about the problem that classifier gives, and it is exploited by the discovery component to lead evolution. Unlike the low fitness, the high fitness gives useful information about the problem, and therefore it should reproduce more through the genetic algorithm.

On each discrete time step, the system receives as input the current state of the environment s and the match set is formed out of classifiers in the population whose condition matches s. Then, the system evaluates the prediction of the actions in the match set, an action a is selected from those in the match set according to certain criterion and set to the environment to be performed. The system receives a reward r according to s and a. The reinforcement component is implanted with the discovery component, the genetic algorithm, randomly selects two classifiers from the population based on their fitness; the genetic algorithm applies crossover and mutation generating two new classifiers (Holmes et al., 2002).

There are three basic genetic operators, selection, crossover, and mutation that recombine the selected condition part of a classifier to make a new classifier for the next steps. The general algorithmic description of the genetic algorithm is as follow (K. Deb, 1999):

- Initialize parameters
- Make the initial population and fitness
- Repeat:

    Selection of parents to produce offspring (Reproduction)

    Crossover

    Mutation

    Updates population and fitness of individuals
- End after meeting the condition

There are some methods such as tournament selection, ranking selection, and proportionate selection to identify good (usually above average) solutions in the population and eliminate the under average resolutions and replace them by copies of good answers.

Crossover operator takes a part of picking two solutions that called parent solutions from the new population that created after selection and exchange between these picking selections. For example, A and B are two parent strings (condition) that are chosen by length five from the population,

$$A = a_1 a_2 a_3 a_4 a_5$$

$$B = b_1 b_2 b_3 b_4 b a_5$$

If the single-point crossover operator, where this is performed randomly choosing a crossing site along the string and by exchanging all bits on the right side of the crossing site, is 3, the resulting strings are two offspring A' and B':

$$A' = b_1 b_2 b_3 a_4 a_5$$

$$B' = a_1 a_2 a_3 b_4 b a_5$$

Mutation operator is applied to make random changes with low probability. This operator can change one bit of string (condition) $0\ to\ 1$ or vice versa. The mutation is to keep diversity in the population.

In the learning classifier systems, GA performs on the population of classifiers. Two classifiers are selected and copied from the population with a probability proportional to their fitness. The crossover operator performs on the selected classifiers from the single-point. Then the mutation performs on the resulting classifiers. The GA produces classifiers with new conditions and new fitness values to be applied as input and make general rules in the learning classifier systems.

The GA in XCS starts by checking the action set to see if the GA should be applied at all. To implement a GA the average period since the last GA application in the set must be greater than the considered threshold. Next, two-parent classifiers are selected by different selection methods such as roulette wheel selection based on fitness and the offspring are created out of them. Then, the offspring are likely crossed and mutated. If the offspring are crossed, their prediction, error and fitness (will be in details explained in the next chapter) values are set to the average of the parent's value. Finally, the offspring are inserted in the population, followed by corresponding deletions (Butz & Wilson, 2002). The RunGA function that written in MATLAB is presented in Appendix.

2.3.3.2 Learning

Learning is a process to build a general model through experiences by interacting with the environment. This concept of learning via reinforcement is a crucial mechanism of the LCS architecture. The terms learning, reinforcement, and credit assignment are often used interchangeably within the literature. Each classifier in the LCS population contains condition, action and one or more parameter values such as fitness associated with it. These parameters can identify useful classifiers in obtaining future rewards and encourage the discovery rules (R. J. Urbanowicz and J. H. Moore, 2009). A learning agent must be able to sense the input from the environment to take actions by considering the goal or goals relating to the state of the environment.

Based on the literature, some essential learning techniques for a learning agent are as follows:

Supervised learning is learning from a labelled training set that provided by a knowledgeable external supervisor. Each example consists of an input object and desired output value. In another word, each case is a description of a situation with a specification of the correct action that the system should take to that situation. The objective of this kind of learning is to generalize a rule for acting correctly in situations not present in the training set.

Unsupervised learning is learning from unlabeled data; this kind of learning is typically about finding a hidden structure in the collection of unlabeled data.

Reinforcement learning is different from supervised and unsupervised learning; in interactive problems, supervised learning is often impractical to obtain proper behaviour which agent has to act. Also, reinforcement learning is trying to maximize a reward signal instead of trying to find a hidden structure. So, reinforcement learning is considered as a third machine learning paradigm, beside the other paradigm as well (Richard S. Sutton, 2017).

2.3.4 Learning classifier systems and eXtended classifier systems

Learning classifier systems is an algorithm to seek a single best-fit model to maximize the achieved environment payoffs. The replacement of the strength parameter by new attributes to classifier systems causes LCSs identifies the paradigm that proposed by Holland. One the most studied and the most applied LCSs is eXtended classifier systems (XCS) (Holmes et al., 2002). The XCS classifier systems were first presented by Stewart Wilson (Wilson, 1987) which is the top of the

research for developing a new LCS. XCS keeps all the main ideas of the previous model while it introduces some fundamental changes. In the next chapter, XCS will be discussed in details.

CHAPTER 3 XCS CLASSIFIER SYSTEM AND ENSEMBLE AVERAGING

This chapter is divided into three parts. In the first part, XCS, the second part, ensemble averaging, methodology and the third part environments are respectively discussed in details.

3.1 XCS classifier system

3.1.1 Introduction

In classical classifier systems, the classifier strength parameter is applied as a predictor of future reward and as the classifier's fitness for the genetic algorithm (GA). Since predicted reward cannot accurately represent fitness, XCS is a developed learning classifier systems (LCS) to overcome the dissatisfaction with the behaviour of classical learning classifier systems. While prediction of expected payoff maintains in each classifier in XCS, the fitness is a separate number base on an inverse function of the classifier's average prediction error, a measure of the accuracy of the prediction (S. W. Wilson, 1995).

Changing the definition of fitness upon the accuracy of a classifier's reward prediction is one of two changes to XCS. The other difference is to execute the genetic algorithm in a niche, means a set of environment states each of which is matched by the same set of classifiers, instead of panmictic (S. W. Wilson, Wilson, Xcs, & System, 1998). If there is a panmictic GA, each classifier has an equal probability of crossing with any other classifier in classifier population [P], classifiers have the same strength (S. W. Wilson, 1994). These changes in XCS shift it to accuracy-based fitness and make it superiority to traditional classifier systems. The result of a combination of accuracy-based fitness and niche GA is maximally general, complete and accurate mapping from input space and actions to payoff predictions, $S \times A \Longrightarrow P$ (S. W. Wilson, 1995).

3.1.2 Description of XCS

Figure 3-1 gives a view of the interaction of XCS which consist of an environment by using detectors for sensory input and effectors for motor actions. Also, the environment at the time offers a scalar reinforcement as a reward (S. W. Wilson, 1995). This figure shows the similarities of classical LCS and XCS in matching sensory information and the condition of classifiers in the population. Selecting an action based on the strategy that is applied by matched classifiers, and

turning back the effect of action to the environment by a payoff to update the population of classifiers and increase the knowledge of the system about the problem.

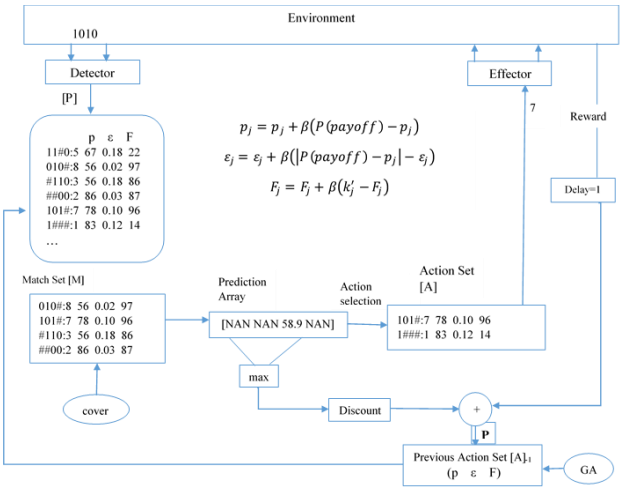

Figure 3-1: A detailed block diagram of XCS

XCS interacts with the environment by detectors to receive information and perform an action by effectors in the environment, and at each time step, it receives a delayed scalar payoff from the environment. In figure 3-1, $[P]$ is the population set that contains the all population of classifiers. Each classifier has two parts condition and action that take place left and right side respectably and are separated by ":". Three values of prediction p, prediction error ϵ and fitness parameter F are associated to each classifier. N shows the maximum size of $[P]$ are randomly generated.

3.1.3 Performance component and a reinforcement component

At each time step, match set $[M]$ is created out of the current population then a prediction system for each action in $[M]$ is formed to propose the executed action (Butz & Wilson, 2002). The task of covering is to generate a classifier with a matching condition and a random action when none of the classifiers in the population match the input (Stewart W, 2007). In the next step, the prediction array $[PA]$ is modified out of a match set to predict the resulting reward for each possible action. Based on $[PA]$, one action is chosen for execution and action set $[A]$ is formed. Then the action with high accuracy is executed and previous action set $[A]_{-1}$ is modified by using the Q-learning-like reward which is a combination of the previous reward and the largest action prediction in $[PA]$.

Each classifier keeps its knowledge about the problem that received from the environment as input by detectors. Generally, each classifier is a condition-action-prediction rule that respectively are

i) $C \in \{0, 1, \#\}^L$ determines the input states, "#" a "don't care" symbol to permit the formation of generalization. This symbol shows no tendency or theoretical reason for accurate generalizations to evolve. L is a number of bit in each situation.

ii) $A \in \{a_1, \dots, a_n\}$ presents the action that the classifiers propose.

iii) p estimates the payoff expected if the classifier matches and its action is taken by the system (Butz & Wilson, 2002).

New classifier parameters consist of prediction p_j, error ε_j, and fitness F_j. The prediction p_j is a statistic estimating the Q- learning-like P when that classifier matches, and its action is chosen by the system. P is updated by adding the discounted maximum of $P(a_i)$ of the prediction array by multiplying discount factor γ where $0 < \gamma < 1$ and previous action time step reward. In the next section, all parameters will be completely explained. Also, the updated parameters will be studied to see the parameter's influence on the algorithm's behavior while holding the rest of the parameters constant. P is used to adjust the p_j, ε_j, and F_j.of the classifiers in $[A]_{-1}$ with learning parameter β. The updating process is as follows:

1. $p_j = p_j + \beta(P(payoff) - p_j)$ where β is learning rate constant.
2. Additional considered parameters in a classifier are each classifier's error (ε) is an estimate error of p_j, it is updated by $\varepsilon_j = \varepsilon_j + \beta(\left|P(payoff) - p_j\right| - \varepsilon_j)$,
3. The fitness (F) that its calculation is a little complicated, a fitness is updated when it is in $[A]_{-1}$. The update of this value depends on the accuracy of the classifier. There are three steps:

   3.1. Calculate classifier's accuracy k_j which based on the current value of ε_j

$$k_j = \begin{cases} \alpha(^{\varepsilon_j}/_{\varepsilon_0})^{-\vartheta} & \varepsilon_j > \varepsilon_0 \\ 1 & otherwise \end{cases}$$

   The threshold determines accurate and lowers accurate classifiers ε_0 and α that are constant to control the shape of accuracy, ϑ is applied in an internal function which scales error nonlinearly (Bull, 2015) (Butz & Wilson, 2002).

   3.2. Calculating relative accuracy k_j' where k_j' is obtained by dividing its accuracy by the total accuracies in the set for each classifer.

3.3. Adjusting the fitness of the classifier $F_j = F_j + \beta(k_j' - F_j)$.

The rest of the components in the population are numerosity (n); when a new classifier is generated, the population of classifiers is check out to search the same classifier of a new one. If any classifier with the same condition and action of the new generated classifier is available or not. If there is not the same classifier, new generated one is added to the population and one is added to numerosity, otherwise, the new one is not added to the population and one is added to the numerosity of classifier. These classifiers are called macro classifiers (S. W. Wilson et al., 1998).

3.1.4 Learning parameters in XCS

Some following parameters are used to control the process of learning:

(N) is the maximum size of the population, start from an empty population, covering occurs at the beginning of a run.

(β) is learning rate for $p, \varepsilon, f \ and \ as$.

$(\alpha, \varepsilon_0, and \ \vartheta)$ are used in calculating the fitness of a classifier. The value of them will be discussed later.

(γ) is the discount factor used in multi-step problems in updating classifier predictions.

(θ_{GA}) is the GA threshold. When the average time since the last GA in the set is greater than θ_{GA}, the GA is applied.

(χ) is the probability of applying crossover in the GA.

(μ) is the probability of mutating an allele in the offspring.

(θ_{del}) is the deletion threshold. The fitness of a classifier may be considered in its probability of deletion if the experience of a classifier is greater than θ_{del}.

(δ) specifies the fraction of the mean fitness in $[P]$ below which the fitness of a classifier may be considered in its probability of deletion.

(θ_{sub}) is the subsumption threshold. The experience of a classifier must be greater than θ_{sub} to be able to assume another classifier.

$(P_{\#})$ is the probability of using a # in one attribute in condition when covering happens.

$(p_I), (\varepsilon_I)$ and (f_I) are used as initial values in new classifiers. They are close to zero.

(p_{explr}) specifies the probability during action selection of choosing the action uniform randomly.

(θ_{mna}) specifies the minimal number of actions that must be present in $[M]$ or covering.

$(doGASubsumption)$ is a Boolean parameter to test offspring for possible logical subsumption by parents.

$(doActionSetsubsumption)$ is a Boolean parameter to be tested for subsuming classifiers(Butz & Wilson, 2002).

Checking the parameter values in the similar experiments is used to parameter setting. The optimal value of population size (N) is highly depend on the complexity of the environment and the number of possible action. The learning rate, (β), could be in the range of 0.1-0.2. The parameters $(\alpha, \varepsilon_0, and\ \vartheta)$, are normally 0.1, 1% and an integer greater than 1. The discount factor (γ) is between 0 and 1, in many problems in the literature is 0.71. The threshold (θ_{GA}) is often in the range 25-50. The GA parameters (χ) and (μ) are in the range 0.5-1 and 0.01-0.05. The deletion threshold (θ_{del}) and (δ) is respectively often taken 20 and 0.1. $(P_{\#})$ could be around 0.33.

3.1.5 Generalization

According to Wilson, generalization means that different situation with equal consequences in the environment would be recognized by lower complexity than the raw environmental data. As it is mentioned in the previous chapter, generalization in LCS means that a classifier can be matched with more than one input vector that received from the environment (Wilson et al., 1998).

A mapping from state and action to the payoff prediction $S \times A \Longrightarrow P$ will be formed in XCS. In the family of learning classifier systems, XCS contains the classifier's fitness that is given by a measure of the prediction's accuracy and also executes the genetic algorithm in niches defined by match sets. This combination accuracy-based fitness and niche GA leads to accurate and maximally general classifiers. The question is that is it possible to stop over general classifiers by basing fitness on accuracy? That is, how possible classifiers would evolve to be general while satisfying the accuracy criteria? An accurate classifier is a classifier with an error less than ε_0 and a maximally general classifier is a classifier that changing any 1 and 0 in its condition to # cause it is inaccurate. The niche environments have the same payoff to within the accuracy criterion, but represent

different inputs to the system; the goal is to put same payoff niches in one class in order to minimize the population's size.

This mechanism is as follows. Consider two classifiers $C1$ and $C2$ with the same action, where $C1$'s condition is more general than $C2$, it means that $C1$'s condition can be generated from $C2$'s only by changing one or more of 0 or 1s condition to #. Also, they have the same ε. Whenever $C1$ and $C2$ are in the same action set, their fitness values will be updated with the same values. Since $C1$ is more general than $C2$, the probability that $C1$ happens in more match sets is higher and is more productive in GA. Consequently, when $C1$ and $C2$ appear in the next step in the same action set, $C1$ will receive more fitness adjusted value resulting through the GA and $C1$ would eventualy displace $C2$ from the population (Wilson, 1995).

The generalization process should continue as long as more general classifiers can be formed without losing accuracy and should stop. The stopping point is controlled by ε_0. So the classifier should evolve as long as they are general and still less than ε_0. There is no theoretical reason for XCS's tendency to evolve accurate, maximally general classifiers. The reason that XCS cannot evolve to accurate generalization is to fail to eliminate over general classifiers, even though their accuracy is low. Elimination depends on the existence of a more accurate competitor classifier in every action set where the over general occurs. When the niches of the environment are distant, the agent cannot change niches as frequently as it needs to evolve an optimal policy. Also, the mechanism of XCS deletion of over general classifiers is very slow (Wilson, 1999).

There are some generalization method such subsumption deletion that reduces population size. In simple word, subsumption is applied to remove the classifiers that cannot add anything to the capability of learning and making a decision of systems. It is helpful to have a smaller final classifier population (Butz & Wilson, 2002). For example, assume the classifier C_2=###1:4 is accurate and maximally general in some environment. If an error is less than ε_0, it is called "accurate" and if accuracy won't be changed by exchanging 1 or 0 to # is called "maximally general". The classifier C_1=##11:4 is also accurate, but it is not maximally general, because it is subsumed by C_2. It is unnecessary evolved classifier that will be eventually deleted by GA (S. W. Wilson et al., 1998).

Two independent subsumptions, doGASubsumption and doActionSetSubsumption, are mentioned here. First one is applied during the genetic algorithm to compare the condition of an offspring

classifier with parents. If an accurate and sufficient experienced parent logically subsumes offspring, the offspring is not added to the population, but the parent's numerosity is incremented. Although subsumption and doActionSetSubsumption have a similar purpose to GA subsumption, they are different and independent. DoActionSetSubsumption takes place in an action set to search the most general, accurate and sufficient experienced to test among the population and to remove the subsumed classifier from the population (Butz & Wilson, 2002).

3.1.6 XCS and RL

In chapter 2, reinforcement learning (RL) is presented in details. In summary, RL deals with the problem of an agent that has to learn to perform a task by interacting with an unknown environment. It means that the agent only knows the current environmental situation s. The agent choose an action among possible action set a based on s and as a consequence of taking action, it receives a reward r, this reward presents how well the agent behaves regarding to problem solution. So, the goal of the agent is to try maximizing the amount of reward received. To reach this goal, the agent computes a value function $Q(s,a)$ which maps state-action pairs into an estimate of expected cumulative future reward. There are two assumption in RL algorithms, $Q(s,a)$ is presented as a lookup table and each state-action pair is visited an infinite times. But it is not possible to visit every stat-action pair infinite time in infinite time because of the memory requirement. This introduces the problem of generalization. How possible to reuse previous experiences in areas of the problem space that are hardly visited to compute an approximation of $Q(s,a)$. Solution is the value function $Q(s,a)$ is approximated by a parameterized function.

It is presented that learning classifier systems (LCSs) are a method of RL to provide an approach to generalization. In LCS, the value function $Q(s,a)$ is presented by a set of condition-action-prediction rules that called classifiers. If condition matches s, classifiers apply the same s and take the same action a and then combine their predictions to provide an estimate of $Q(s,a)$. If the population of maximally general classifiers are applied in as many situations as possible while estimating $Q(s,a)$, generalization can be obtained (Lanzi & Loiacono, 2006).

Since XCS has a stronger relation to RL than other models in using a modification of Q-learning, the relation between RL and LCS has been mainly focused on Wilson's XCS (Lanzi & Loiacono, 2006).

Q-learning directly adapts XCS's learning algorithm. It says that the prediction of each classifier in the action set is updated by the current reward plus the discounted value of the maximum system prediction on the next time-step. The system prediction for a particular action is based on the fitness-weighted average of predictions of each classifier in the match set that advocate action. For each action, the maximum system prediction over the match set is the maximum of the system prediction. The fundamental difference between XCS and Q-learning is that classifier (rule) predictions are updated from the prediction of other rules. While in Q-learning, the action-value prediction is updated by other action-values. XCS's memory is contained rule sets, whereas Q-learning's memory keeps a table with one entry for each state-action pair (Wilson, 1999).

By applying #s and introducing a mutation in XCS, it will typically reach the same level as a tabular Q-learning and take longer due to errors as accurate general classifiers are discovered and emphasized. XCS can be presented the problem with categorical regularities, so generalization will result in a population that is smaller than the corresponding Q-table. It means that XCS empirically coverages like Q-learning but with generalization (Wilson, 1999).

3.1.7 A literature review on XCS

This section reviews some approaches to solve the over the general problem and developments in the XCS classifier system. In these studies, some component is added or changed, or some different kinds of techniques are applied in XCS.

First, Lanzi identifies that because of the inequality in the exploration of all states in the grid-like environments, XCS has difficulty in getting optimum generalization within some environments. He uses some particular exploration policies and operators to improve the performance of XCS by visiting all the areas of the environment efficiently and frequently and controlling over general classifiers (Lanzi, n.d.).

To solve this problem, Barry gives a modification to the error computation within XCS, which is this calculation tries to make the error measure independent of the position of the classifier in action set to reward. Although the primary results show that this modification can enable XCS to map more extended delayed-reward environments, no further investigation is reported (Barry, 2003).

Later, Butz et al. analyze the similarities between tabular Q-learning, Q-learning with gradient descent, and XCS. They argue that XCS should be more similar to Q-learning with gradient descent

than plain tabular Q-learning because of generalization capabilities of XCS. So, they add gradient descent to the update of the classifier prediction in XCS, which called XCSG, to improve the learning abilities of XCS. Their experiments show that this extension can make XCS more robust and efficient in solving difficult problems (Butz, Goldberg, & Lanzi, 2005).

Zhaoxianng et al. apply R-learning and average reward as the reinforcement learning by XCS that named XCSAR, to replace Q-learning. Their modifications result in some improvement to solve some grid environment and effectively prevent the occurrence of overgeneralization (Zang et al., 2013). A summary of noted LCS and XCS classifier are discussed by (R. J. Urbanowicz and J. H. Moore, 2009)

3.2 Ensemble averaging in machine learning

In supervised learning, the input space is divided into a set of subspaces to distribute the learning tasks between numbers of experts to simplify the computations. A committee machine is a combination of these experts that do the computational simplicities. One of the significant categories of committee machine is the static structure that includes the ensemble averaging method where the outputs of different predictors are linearly combined to have a general output.

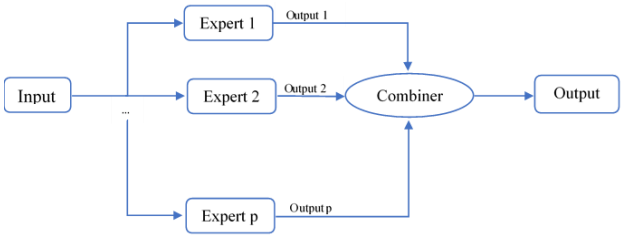

Figure 3-2: Block diagram of a committee machine based on ensemble averaging

The figure 3-2 shows the block diagram of a committee machine based on ensemble averaging method where all individual outputs of different predictors for a typical input are combined to produce an overall output. In any event, by using a committee machine like figure 3-2, the expectation is that overall performance is improved by combining the outputs. The bias-variance dilemma is presented to show this improvement in a combination of experts. The outputs of the experts are assumed to be scalar-valued to simplify the presentation.

Let x denote an unseen input vector and d show the corresponding desired response, x and d represent the realization of the random vector X and random variable D, respectively and $F(x)$

denotes the input-output function realized by the network. The mean-square error between $F(x)$ and the conditional expectation $E(D|X=x)$ is decomposed into its bias and variance component over the space $\mathcal{D}$, which defined as the space encompassing the distribution of all inputs, desired outputs and the distribution of all initial conditions, as follows:

$$E_{\mathcal{D}}[F(x) - E(D|X=x)^2] = B_{\mathcal{D}}\big(F(x)\big) + V_{\mathcal{D}}(F(x)) \qquad (3\text{-}1)$$

where $B_{\mathcal{D}}\big(F(x)\big)$ is the bias squared:

$$B_{\mathcal{D}}\big(F(x)\big) = (E_{\mathcal{D}}[F(x)] - E(D|X=x))^2 \qquad (3\text{-}2)$$

and $V_{\mathcal{D}}(F(x))$ is the variance:

$$V_{\mathcal{D}}\big(F(x)\big) = E_{\mathcal{D}}[(F(x) - E_{\mathcal{D}}[F(x)])^2] \qquad (3\text{-}3)$$

Here a simple ensemble average for the combiner at the output of the committee machine in figure 3-2 is applied. Assume the space of all initial conditions is denoted by $\mathcal{J}$ and $F_I(x)$ is the average of the input-output function of the expert networks in figure 3-2 over a representative number of initial conditions. Based on equation 3-1:

$$E_{\mathcal{J}}[F_I(x) - E(D|X=x)^2] = B_{\mathcal{J}}\big(F(x)\big) + V_{\mathcal{J}}(F(x)) \qquad (3\text{-}4)$$

where $B_{\mathcal{J}}\big(F(x)\big)$ is the squared bias defined over the space $\mathcal{J}$:

$$B_{\mathcal{J}}\big(F(x)\big) = (E_{\mathcal{J}}[F_I(x)] - E(D|X=x))^2 \qquad (3\text{-}5)$$

and $V_{\mathcal{J}}(F(x))$ is the corresponding variance:

$$V_{\mathcal{J}}\big(F(x)\big) = E_{\mathcal{J}}[(F_I(x) - E_{\mathcal{J}}[F(x)])^2] \qquad (3\text{-}6)$$

The expectation of $E_{\mathcal{J}}$ is adapted the space $\mathcal{J}$.

Based on the definition of space $\mathcal{D}$, it is aggregate of the space of initial conditions $\mathcal{J}$ and the remnant space that denoted by $\mathcal{D}'$. The equation (3-1) for $\mathcal{D}'$ is as follow:

$$E_{\mathcal{D}'}[F_I(x) - E(D|X=x)^2] = B_{\mathcal{D}'}\big(F_I(x)\big) + V_{\mathcal{D}'}(F_I(x)) \qquad (3\text{-}7)$$

where $B_{\mathcal{D}'}\big(F_I(x)\big)$ is the squared bias defined over the remnant space $\mathcal{D}'$:

$$B_{\mathcal{D}'}\big(F_I(x)\big) = (E_{\mathcal{D}'}[F_I(x)] - E(D|X=x))^2 \qquad (3\text{-}8)$$

and $V_{\mathcal{D}'}(F_I(x))$ is the corresponding variance:

$$V_{\mathcal{D}'}(F_I(x)) = E_{\mathcal{D}'}[(F_I(x) - E_{\mathcal{D}'}[F_I(x)])^2] \qquad (3\text{-}9)$$

From the definition of spaces $\mathcal{D}$, $\mathcal{J}$, and $\mathcal{D}'$:

$$E_{\mathcal{D}'}[F_I(x)] = E_{\mathcal{D}}[F(x)] \qquad (3\text{-}10)$$

So the equation (3-8) is written in a similar form:

$$B_{\mathcal{D}'}\big(F_I(x)\big) = \big(E_D[F_I(x)] - E(D|X = x)\big)^2 \qquad (3\text{-}11)$$

$$= B_D\big(F_I(x)\big)$$

Since the variance of a random variable is equal to the mean-square value of that random variable minus its bias squared, so

$$V_{\mathcal{D}'}(F_I(x)) = E_{\mathcal{D}'}[(F_I(x))^2] - (E_{\mathcal{D}'}[F_I(x)])^2 \qquad (3\text{-}12)$$

$$= E_{\mathcal{D}'}[\big(F_I(x)\big)^2] - (E_D[F_I(x)])^2$$

Similarly for equation

$$V_{\mathcal{D}}\big(F(x)\big) = E_{\mathcal{D}}[(F(x))^2] - E_{\mathcal{D}}[F(x)])^2 \qquad (3\text{-}13)$$

The mean-square value of the function $F(x)$ over the entire space, $\mathcal{D}$ is intended to be equal to or greater than the mean-square value of the ensemble-averaged function $F_I(x)$ over the remnant space $\mathcal{D}'$. It means

$$E_{\mathcal{D}}[F(x)^2] \geq E_{\mathcal{D}'}[(F_I(x))^2] \qquad (3\text{-}14)$$

Comparison of equation (3-12) and (3-14) shows that

$$V_{\mathcal{D}'}(F_I(x)) \leq V_{\mathcal{D}}\big(F(x)\big) \qquad (3\text{-}15)$$

So, two conclusions are drawn from (3-11) and (3-15), the bias of the ensemble-averaged function $F_I(x)$ related to the committee machine like figure (3-2) is exactly the same as that of the function $F(x)$ related to a single expert. And the variance of the ensemble-averaged function $F_I(x)$ is less than that of the function $F(x)$.

The theoretical findings point that the variance of the ensemble-averaged function is reduced by ensemble averaging the experts over the initial conditions while the bias of the ensemble average function is leaving unchanged (S. Haykin, 1999).

3.2.1 Linear combination of experts

As it is mentioned in the first chapter, combining estimates and methods are not new. Laplace considered combining regression coefficient estimates many years ago. The increasing accuracy by combining multiple methods in a different domain such as statistics, econometrics or extrapolation has been resulted (Clemen, 1989).

Clemen provides considerable literature regarding the combination of forecasts over the years. The conclusion of this accumulated literature is that forecast accuracy can be significantly improved through the combination of multiple individual forecasts. Also, simple combination methods often work reasonably well for more relative complex combinations (Clemen, 1989).

A linear combination of the outputs from all experts returns a scalar output. Suppose x is the common input for all the experts and $y_j(x)$ is output for p experts. The overall output can be defined as:

$$\tilde{y}(x, \alpha) = \sum_{j=1}^{p} \alpha_j y_j(x) \qquad (3\text{-}1)$$

where α is a set of weights. The problem is to find good values for the combination-weights $\alpha_1, \alpha_2, \ldots, \alpha_p$. One approach which is widely used is to apply equal combination-weights, it means a simple average. The simple average is straightforward but assumes that all the estimators are equally good (S. Haykin, 1999).

3.2.2 Linear combination of R- learning and Q- learning

R-learning and Q-learning as famous reinforcement learning algorithm have been introduced in chapter 2, both, R-learning and Q-learning, have a similar form but they have a different meaning (Zang et al., 2013). In this part, the quality of mixing the linear combination of R-learning and Q-learning precisely explain. It is necessary to focus on the methods to compute the estimation of $\widehat{R}(s_t, a_t)$ and $\hat{Q}(s_t, a_t)$ to mix the linear combination of these methods to XCS

$$\hat{R}(s_t, a_t) = r_{imm}(s_t, a_t) - \rho + \max_{a \in A} R(s_t, a) \qquad (3\text{-}2)$$

$$\hat{Q}(s_t, a_t) = r_{imm}(s_t, a_t) + \gamma \max_{a \in A} R(s_t, a) \qquad (3\text{-}3)$$

The linear combination payoff as the correspondence between the system prediction and the action value, payoff P is defined as:

$$P_t = \frac{\delta*(r_{imm}(s_t,a_t)-\rho+\max_{a\in A} R(s_t,a))}{2} + \frac{\theta*(r_{imm}(s_t,a_t)+\gamma\max_{a\in A} R(s_t,a))}{2} \quad where\ \delta = \theta = 1 \qquad (3\text{-}4)$$

Also, the maximum reward of these methods is considered. In a mathematical word:

$$P_t = \max((r_{imm}(s_t,a_t) - \rho + \max_{a\in A} R(s_t,a)), (r_{imm}(s_t,a_t) + \gamma\max_{a\in A} R(s_t,a))) \qquad (3\text{-}5)$$

To update the average reward ρ is calculated by

$$\rho = \rho + \beta_\rho \left(r_t + \max_{a\in A} PA(s_t,a) - \max_{a\in A} PA(s_{t-1},a) - \rho\right) \qquad (3\text{-}6)$$

The computation of ρ is just located before the computation of the combination of R-learning and Q-learning. The learning rate follows the simple rule as follow:

$$\beta_\rho = \beta_\rho - \frac{\beta_\rho^{max} - \beta_\rho^{min}}{Numberoftrial} \qquad (3\text{-}7)$$

Where $\beta_\rho^{max} = 0.005$ and $\beta_\rho^{min} = 0.0005$. The initial value of β_ρ is β_ρ^{max}. $Numberoftrial$ is the number of the problems in an experiment. β_ρ is updated at the beginning of each exploration (Zang et al., 2013).

In the following table, the summary of all named methods is depicted:

Table 3-1: The review of learning methods

Name	Description
XCSG	The extension of XCS to gradient-based update methods, Q-learning with gradient descent $$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)]$$ $$0 \leq \alpha \leq 1$$
AXCS	The modification to XCS to maximize the average rewards $$R_{t+1}(s_t, a_t) = (1-\beta_R)R_t(s_t, a_t) + \beta_R[r_{t+1} + \max_a R_t(s_{t+1}, a) - \rho]$$ $$\rho = (1-\beta_\rho)\rho + \beta_\rho r_{t+1}$$ $$0 \leq \beta_R, \beta_\rho \leq 1$$
XCSAR	Use of average reward in XCS by an undiscounted reinforcement learning technique called R-learning $$R_{t+1}(s_t, a_t) = (1-\beta_R)R_t(s_t, a_t) + \beta_R[r_{t+1} + \max_a R_t(s_{t+1}, a) - \rho]$$ $$\rho = (1-\beta_\rho)\rho + \beta_\rho(r_{t+1} + \max_{a\in A} R_{t+1}(s_{t+1}, a) - \max_{a\in A} R_t(s_t, a))$$ $$0 \leq \beta_R, \beta_\rho \leq 1$$
MIXCS	Linear combination of XCSAR and XCSG, a linear mix of discounted reward based on Q-learning and undiscounted reward based on R-learning $$P_t = \frac{\delta * (r_{imm}(s_t, a_t) - \rho + \max_{a\in A} R(s_t, a))}{2} + \frac{\theta * (r_{imm}(s_t, a_t) + \gamma \max_{a\in A} R(s_t, a))}{2}$$ $$where\ \delta = \theta = 1$$

3.3 Environments

Two two-dimensional grids are defined as the environment to examine the presented methods. One of these environments is a 5×5 cell grid where every cell contains different objects. This environment is inspired by a Woods1 environment that is presented by Wilson to apply the basic classifier systems that are called Environment1. As figure 3-3 shows 5×5 cell grid presents 16 opportunity situations $(.)$ to trade and $eight$ situations to hold (O), and the maximum profit will be made in (F) cell that are provided in 25 cells.

```
. . . . .
. OOF .
.OOO.
.OOO.
. . . . .
```

Figure 3-3: Environment1

Another environment is a 9×9 cell grid where contains 36 cell opportunity situations $(.)$ for trading, 44 cases to hold (O), and the maximum profit will be made in (F) cell that are only provided in 1 cell. This environment is inspired of Maze6 and it is called Environment2.

```
OOOOOOOOO
O . . . . . O F O
O . . O.O O*O
O  . O . . . . . O
O . . . O O. . O
O . O.O . . O O
O  . O . . . . .  O
O  . . . . .O .  O
OOOOOOOOO
```

Figure 3-4: Environment2

As it is mentioned in chapter 2, the agent can sense $eight$ cells around it and store this situation in a vector to help to choose the next action at each time step. Every component (F), $(.)$ and (O) are defined 2 bits as $11, 01$ and 00 respectively. So, a state is defined by 16 bits in a vector that is called detector vector at each time step. As an example, the detector vector of the agent in figure 3-4 according to its situation is '1101010100000101'. In the previous part, it is mentioned that the whole knowledge of the agent is stored in the population set. It contains all classifiers. Each classifier contains a condition that is made of state and an action part. In the case of trading first and second 8 characters of the condition are considered buy and sell respectively. So, first 4 possible actions are reserved for buying and second 4 ones are consider for selling. The goal is to

predict the price movement of tomorrow stock price by using its available input information set; reward function is to assign 0 for incorrect prediction and a positive value 1000 for the correct prediction that suited in (F) cell. In the next chapter, the presented techniques will be applied in these environments to see the results.

3.3.1 AlphaGo

This chapter will be finished by introducing the first computer Go program that beat a human professional Go player. Game of Go is a game can be played on a square board conventionally marked 9×9, 13×13, and 19×19 of crossing lines. It is an abstract strategy means that does not rely on a theme (Thompson, J. Mark, 2000). This game is the challenging of classic games for artificial intelligence because of its huge search space and the difficulties of evaluating board positions and moves. A computer program that is called AlphaGo is introduced to play the board game of Go in different versions.

This new approach applies “value networks” to evaluate board position and “policy network” to choose moves. A novel combination of supervised learning from human expert game and reinforcement learning from the game of self-played is applied to train these deep neural networks. AlphaGo uses a Monte Carlo tree search algorithm that simulates thousands of random games of self-play, to find its moves based on previous knowledge learned. The neural networks play the game of Go at the level of state-of-the-art Mont Carlo tree search program (Silver et al., 2016).

CHAPTER 4 EXPERIMENTS AND RESULTS

As it is mentioned in the previous chapter whole knowledge of the agent is stored within the population set. In a first step, the agent is randomly located and then starts to explore the environment in our case the stock market in order to investigate the stock market to learn the rules for the price prediction in the next step. The prediction of a future stock price is crucial to the decision making, i.e. to buy or to sell a certain stock. The prediction quality is evaluated based on the rewards received.

4.1 Environment1

First, the optimal average path to the cell (F) in this environment is calculated with all trade opportunities located in the left, right, top and bottom edge of the environment. The optimal path of each side is:

$$right: 1 + 1 + 1 + 2 + 3 +$$

$$bottom: 3 + 4 + 5 + 6 +$$

$$left: 3 + 4 + 5 +$$

$$top: 1 + 1 + 2 + 3$$

For the given optimal path, the average number of moves to reach maximum profit (i.e. Food cell) is approximately 2.5.

The following table summarizes the information obtained by the agent during the experiment. All these actions reached to goal to correctly predict the price movement of tomorrow's stock price.

Table 4-1 : Macro classifiers from the experiments

Condition	Action	Prediction P	Error ε	Fitness F	Numerosity n
#00000#0########	7	1000	0	0.2	1
000#0#0011#00###	5	1000	0	0.8	1
00000#0011#00###	5	1000	0	0.46	1
#00#000011#00###	5	1000	0	0.1	1
000#000011#00###	5	1000	0	0.1	1
0#0000#0########	8	1000	0	0.1	1
00#00#1####0####	4	1000	0	0.7	1

As it is mentioned in chapter 2, the possible action set is divided into buying or selling based on the first and last eight characters of detector description. In the following table, the numerosity for each possible action is shown where an agent would always take the same action for a similar set of conditions. Table 4-1 shows the conditions of the set for action 5 which is part of the selling actions. Table 4-2 depicts the number that each action is chosen for similar condition set. Since # helps to generalize the conditions, it can also be used to generalize a rule.

Table 4-2: Numerosity for each possible action

Buy				Sell			
Act. 1	Act. 2	Act. 3	Act. 4	Act. 5	Act. 6	Act. 7	Act. 8
30	18	32	1101	1139	1636	186	856

In this market, the maximum profit will be reached by selling the stock. In the following, the performance of applying MIXCS in Environment1 is discussed.

Figure 4.1 shows the performance of the linear combination of two methods, Q-learning and R-learning MIXCS for 2000 iteration. The minimum and a maximum number of steps to Food cell are 0 and 14, with zero meaning that the agent is randomly located in the food location.

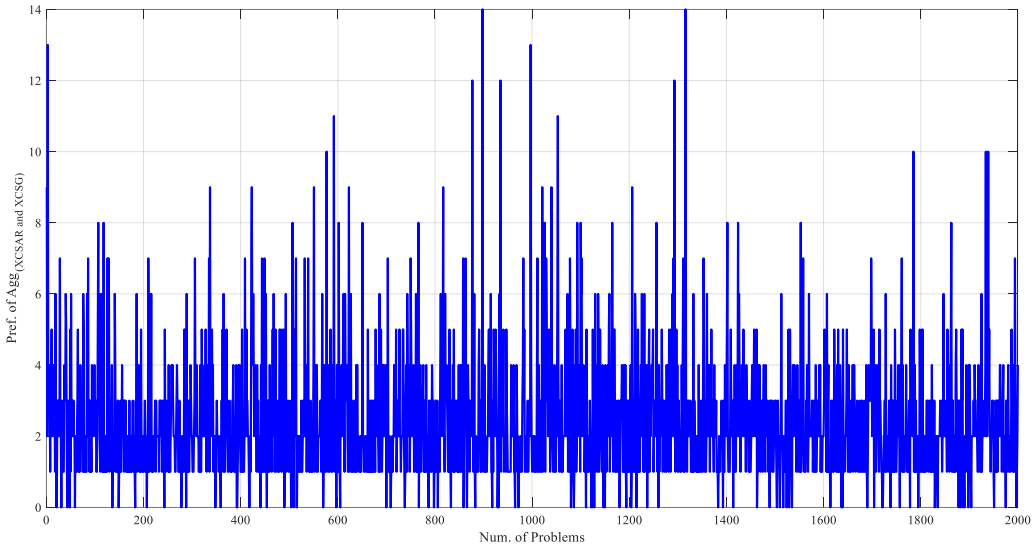

Figure 4-1: Performance of applying MIXCS to Environment1

The estimate of average rewards in XCSAR, XCSG and MIXCS is shown in figure 4.2. The optimum average profit is 400, the average reward of MIXCS is 585.7, while the average reward of XCSAR is higher than 585.7 (the blue curve) and the average reward of XCSG is less than 585.7 (the red curve). Assuming a linear combination of XCSAR and XCSG in Environment1, it can be seen from figure 4.2 that the performance is closer to the optimum average profit.

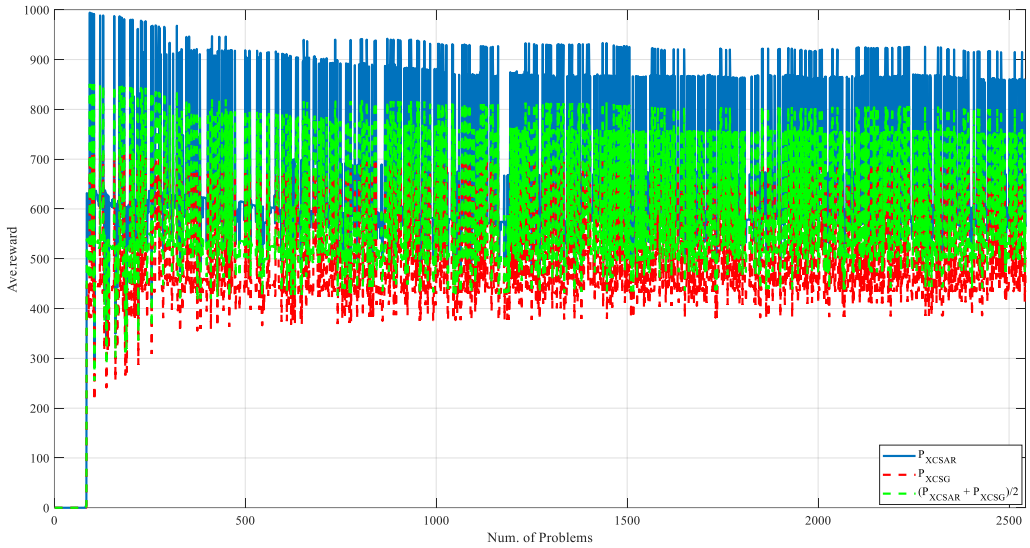

Figure 4-2: Average reward of applying MIXCS to Environment1.

From figure 4.3 it can be seen that no generalization is obtained when XCSAR and XCSG are associated while the combination of XCSAR and XCSG will converge after some iterations. The trend of population size is incremental.

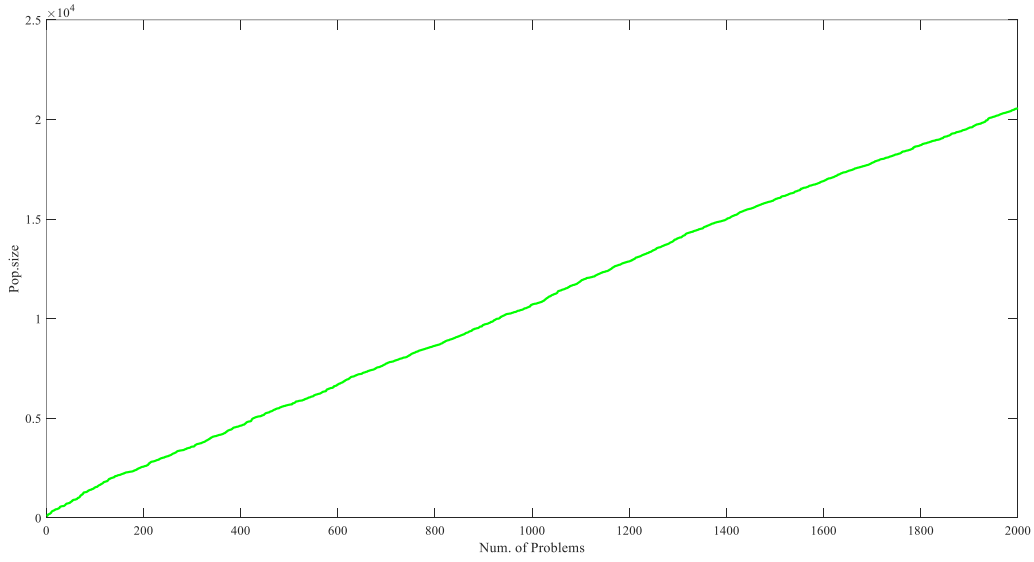

Figure 4-3: Population size in macro classifier for applying MIXCS to Environment1.

In the other test, the performance of comparison prediction of the tomorrow stock price (prediction of next movement) between the two methods is shown.

Minimum and maximum number of step to Food cell are 0 and 93 respectively. Since the agent is randomly put in location, the minimum steps to Food cell can be 0 which means that the agent is directly set in Food cell location. As can be seen from the figure, the minimum number of steps to Food cell is already reached during the first iterations. Having minimum steps in first iterations show that the code is not stable within for this number of iterations and that more iterative cycles might be required.

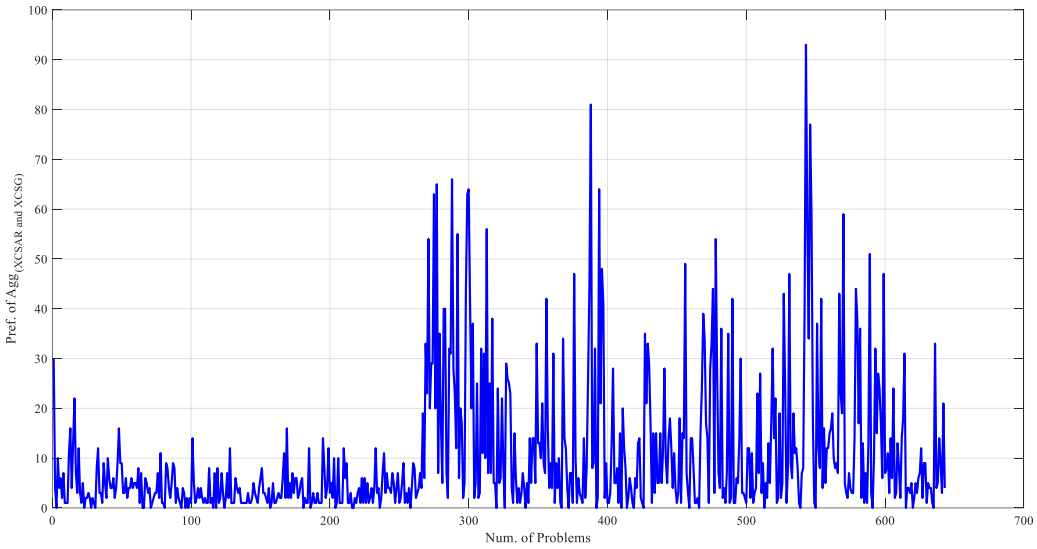

Figure 4-4: Performance of applying MIXCS to Environment1

As figure 4.2 shows the predicted average profit for XCSAR is greater for XCSG, so in the present comparison, the computed average reward obtained by the R-learning method is selected.

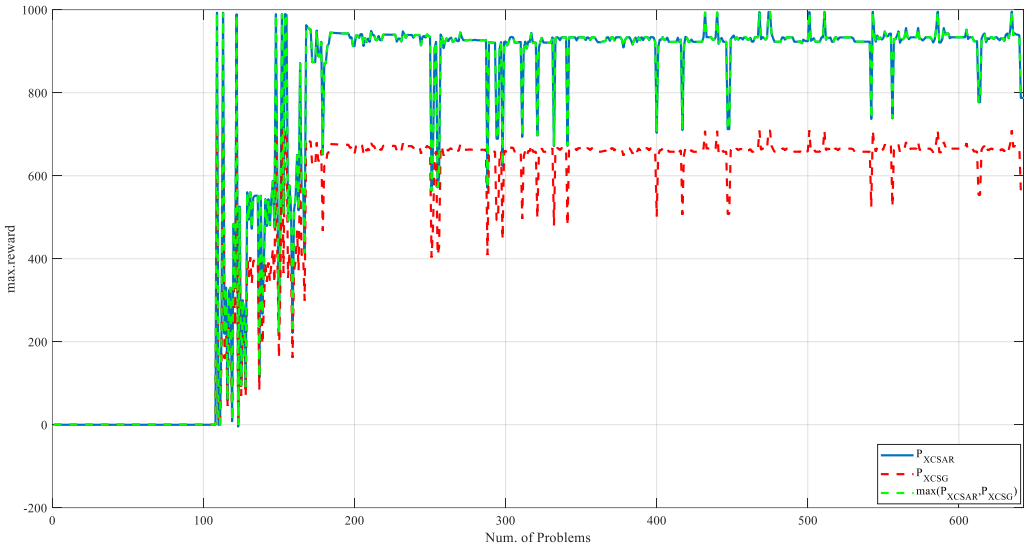

Figure 4-5: Average reward approach, discounted reward approach and maximum of these approaches applied to Environment1.

Figure 4.6 shows the number of classifiers as it increases over time so that after 643 iterations no generalization is obtained. In this case, the agent needs to continue to explore finding a general rule for the decision making and hence the reliable actions.

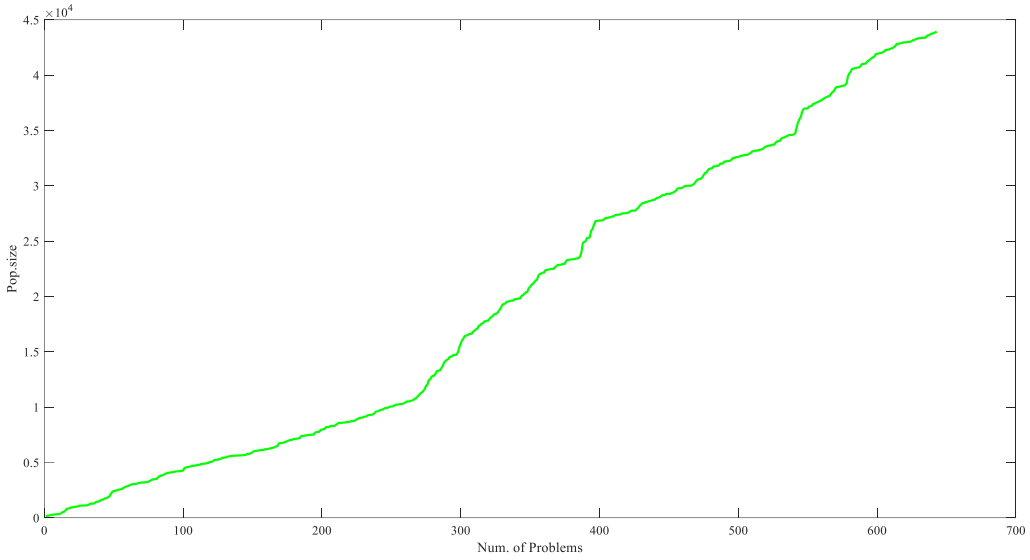

Figure 4-6: Population size in macro classifier for applying the max of XCSAR and XCSG to Environment1

4.2 Environment2

In this part, the results of using the association of XCSAR and XCSG in Environment2 are discussed. The located rules as obstacles, reward and the size of Environment2 are utterly different to Environment1.

The optimal average path to Food cell in Environment2 is

$$line2: 7 + 6 + 5 + 5 + 5 +$$

$$line3: 7 + 6 + 4 + 1 +$$

$$line4: 7 + 5 + 4 + 3 + 2 + 2 +$$

$$line5: 7 + 6 + 5 + 3 + 3 +$$

$$line6: 7 + 6 + 4 + 4 +$$

$$line7: 8 + 6 + 5 + 5 + 5 + 5 +$$

$$line8: 8 + 7 + 6 + 6 + 6 + 6 +$$

There are 36 blank points in this environment. It reveals that the optimal average movement to Food cell, the goal of the agent, in this environment is 5.4 steps.

The numerosity for each possible action for Environment2 is listed in table 4.3,

Table 4-3: Numerosity for each possible action

Buy				Sell			
Act. 1	Act. 2	Act. 3	Act. 4	Act. 5	Act. 6	Act. 7	Act. 8
188	17	32	1089	1120	1555	202	849

In this market condition, the maximum profit will be reached by selling the stock. In the following, the performance of applying MIXCS in Environment2 is discussed.

Although the number of classifiers increases in each time step, the movement of the agent is limited to 98 iterations in this environment. It shows that it needs more information to predict the price for the next steps. This information could be reached by interacting with other agents.

The population size in MIXCS is illustrated in 98 iterations. Since the agent is randomly located in the environment, the minimum number of steps is 0, and the maximum number of the steps is 219 for 36 blank points.

The performance of the investigated maximum reward approach and population size of each iteration are shown in figure 4-7,

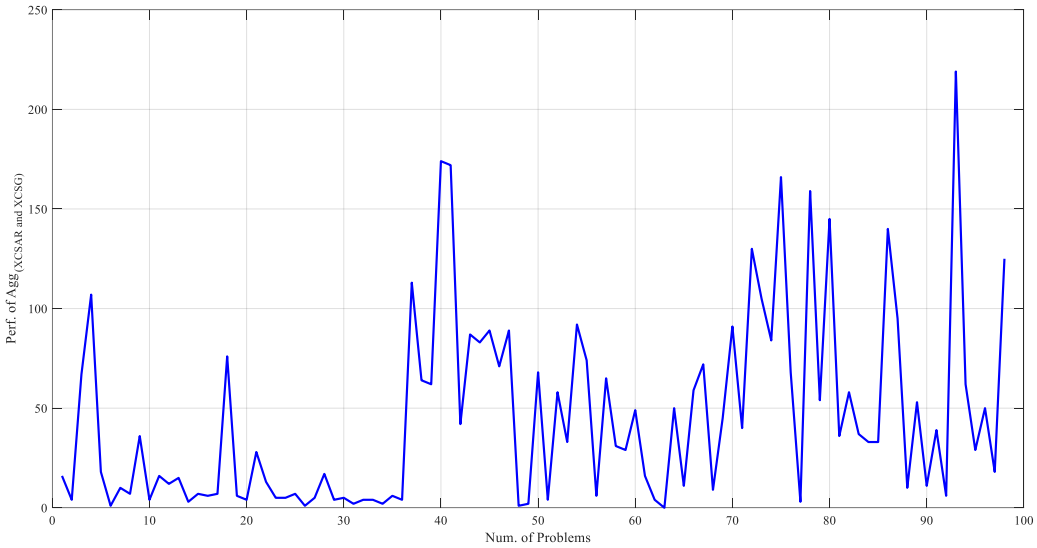

Figure 4-7: Performance of applying MIXCS to Environment2

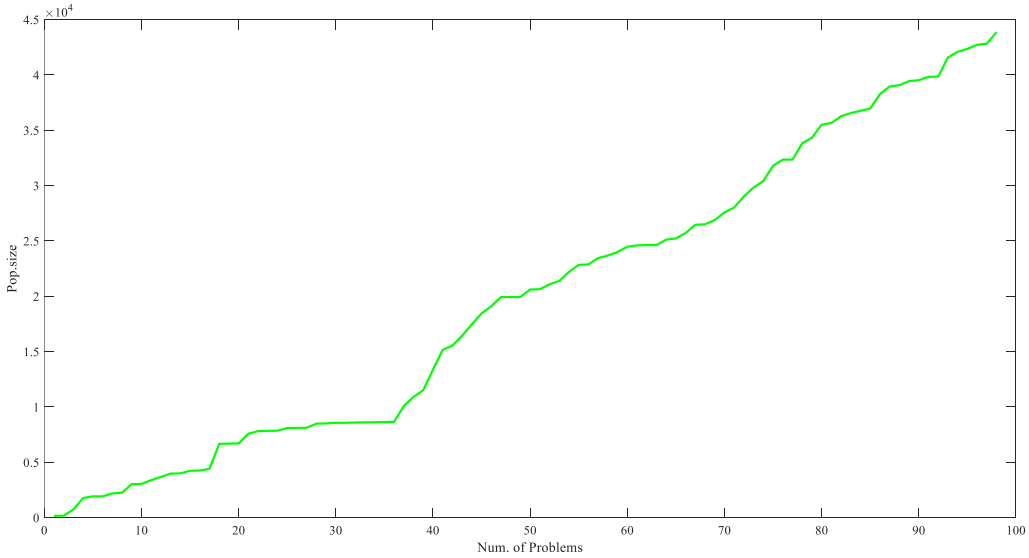

Figure 4-8: Population size in macro classifier for applying MIXCS to Environment2.

In the same way, the estimate of the computed average rewards in XCSAR, XCSG and MIXCS, are presented in figure 4.9. The reward of reaching the Food cell state is considered 1000, and the average number of movement to Food cell in this environment is 5.4 which leads to an optimum

average profit is $\frac{1000}{5.4} = 185{,}19$. It can be readily seen from the figure, that all techniques overestimate the average profit.

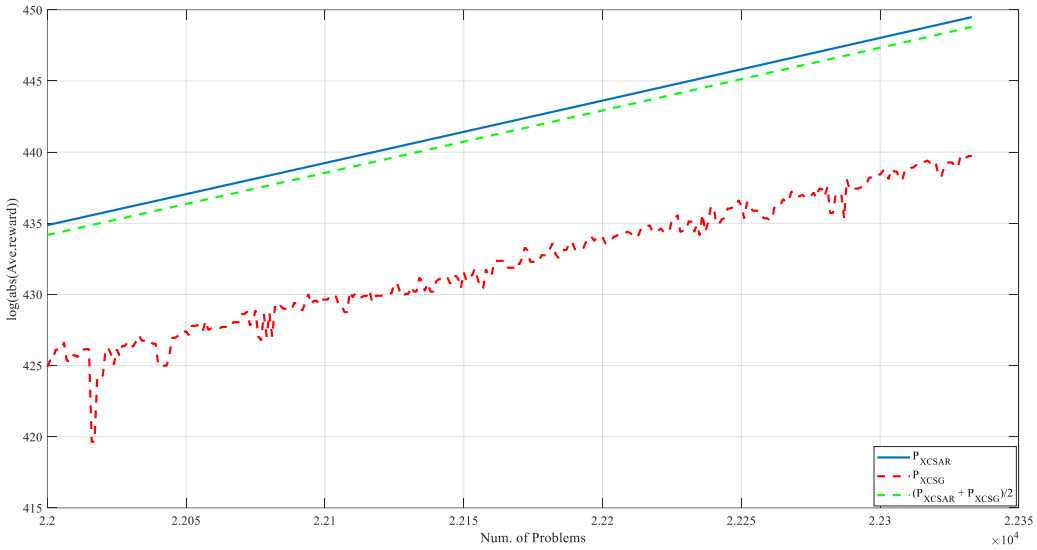

Figure 4-9: Average reward of applying MIXCS to Environment2.

The comparison of the tomorrow stock price (prediction of next movement) for two methods and the above population size is based on18 iterations. The minimum and maximum steps to reach food in this environment are 1 and 249 respectively. It was shown that the population size increases incrementally over the iterations with a minimum of 1576 classifiers during the first step. A generalized rule was not obtained in this environment.

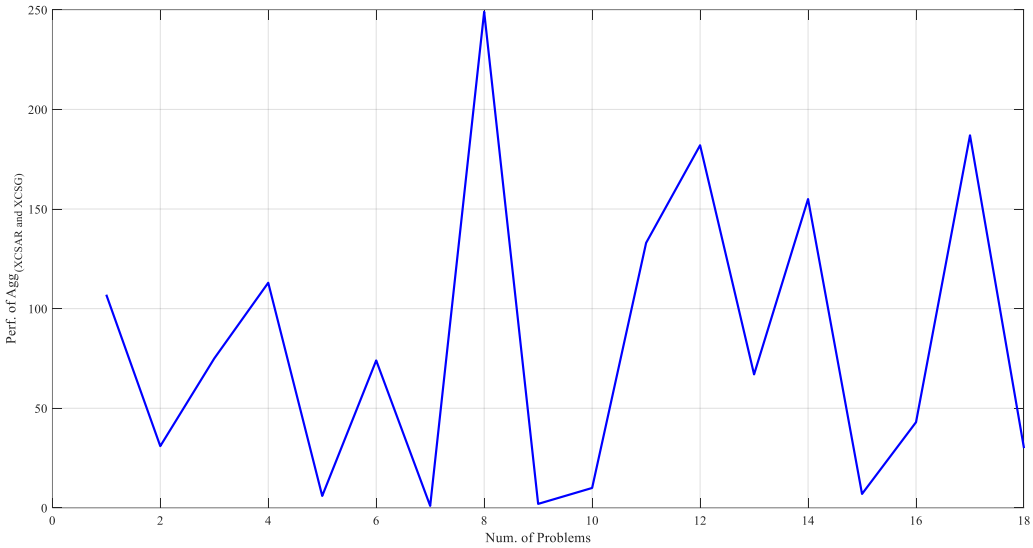

Figure 4-7: Performance of applying MIXCS to Environment2

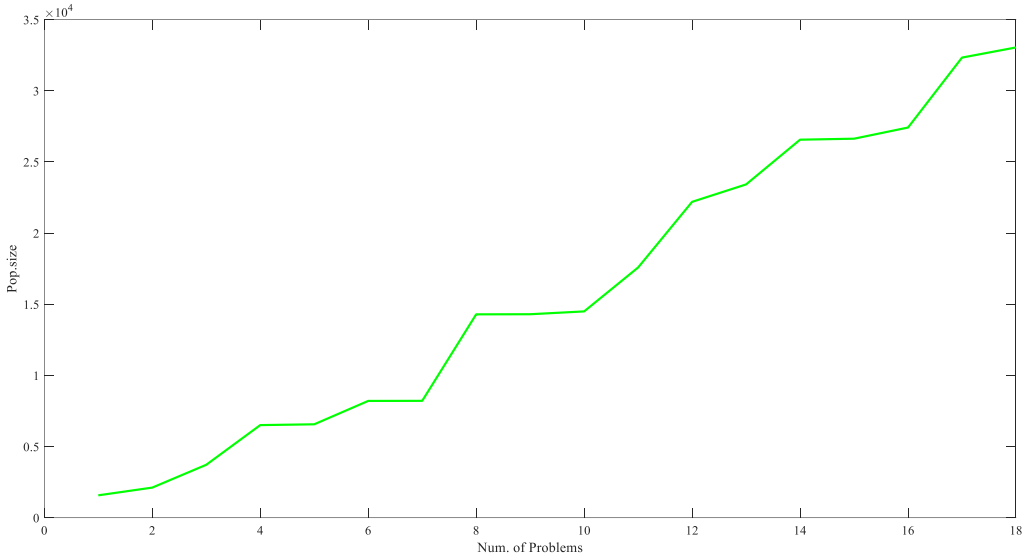

Figure 4-8: Population size in macro classifier for applying the competition of XCSAR and XCSG to Environment2.

Figure 4.12 demonstrates average prediction for XCSAR, XCSG and their comparison of them. As it is depicted in the following graph, XCSAR approaches the optimal value closer than XCSG. Moreover, should be chosen in this case.

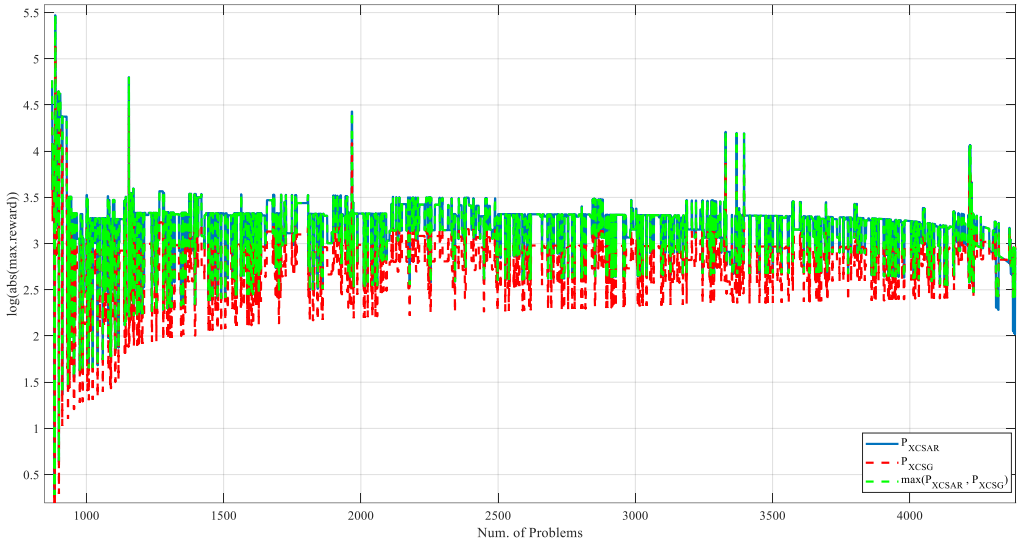

Figure 4-9: Average reward approach, discounted reward approach and maximum of these approaches applied to Environment2.

Environment2 is not only bigger than Environment1 but also the assumed rules, i.e. the obstacles and reward, have a more complex definition. As a consequence, an increase in reliability for the predictions requires not only forecasting through multiple models, but also the interacting with other agents can improve the applicability of the market information simulated.

CHAPTER 5 CONCLUSION AND FUTURE WORK

We have proposed a combination of simple techniques that are applied to learn a policy to maximize the discounted and undiscounted sum of future profits that the environment received. The proposed technique is applied to calculate payoff, in the market case profit, that is based on the predictions contained in the prediction array. This value is used to adjust prediction, fitness and prediction error. The combination of two learning techniques is applied in two two-dimensional grid that is called Environment1 and Environment2 that are inspired by Woods1 and Maze6 respectively. Environment2 is more complicated and bigger than Environment1 concerning the obstacles and food cells. In the case of a trade market, the possible action set is divided into buying and selling actions. On a more abstract level, these environments represent markets with specific rules from which an agent learns rules through exploring to find a policy based on the knowledge of the system. To maximize the profit, the agent should predict the profit of each movement and update its knowledge at each time step.

The performances of this combination that is called MIXCS have been investigated in two Environment1, and Environment2 and the estimated average profits of the three methods are compared with each other. The results show that the calculated average reward of a combination of single techniques, MIXCS, is close to the optimum average profit compared to the use of two single techniques combined for Environment1. The same approach applied to Environment2 did not yield the same the result due to the differences between environments. Based on the possible action set and division on this set into buying and selling, and the numerosity, similar actions in the same set of condition received, for each action, the agent has made a profit by selling in both considered environments as a market.

5.1 Future work

This work can be further expanded in the multiplexer problem environment for the stock markets where the classifiers are rewarded by any changes instead of a fixed number in a determined location.

Besides the interaction with the environment, considering a multi-agent approach can help to show the interaction of agents with each other to take the action that has the highest fitness in prediction array.

Furthermore, applying other learning technique such as H-learning and combining with R-learning as two undiscounted reward systems to examine the performance could be suggested.

BIBLIOGRAPHY

Barry, A. (2003). Limits in Long Path Learning with XCS, 12.

Bull, L. (2015). A brief history of learning classifier systems: from CS-1 to XCS and its variants. *Evolutionary Intelligence*, *8*(2–3), 55–70.

Busoniu, Lucian, et al. (2010). *Reinforcement learning and dynamic programming using function approximators.* CRC press.

Butz, M. V., Goldberg, D. E., & Lanzi, P. L. (2005). Gradient Descent Methods in Learning Classifier Systems: Improving XCS Performance in Multistep Problems. *IEEE Transactions on Evolutionary Computation*, *9*, 452–473.

Butz, M. V., & Wilson, S. W. (2002). An algorithmic description of XCS. *Soft Computing*, *6*(3–4), 144–153.

Clemen, R. T. (1989). Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, *5*, 559–583.

Hohendorff, J. M. (2005). An Introduction to Markov Chain Monte Carlo, 39.

Holland, J. H., & Miller, J. H. (1991). Artificial Adaptive Agents in Economic Theory, 7.

Holmes, J. H., Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (2002). Learning classifier systems: New models, successful applications. *Information Processing Letters*, *82*, 23–30.

K. Deb, K. (1999). An introduction to genetic algorithms, *24*, 293–315.

Lanzi, P. L. (n.d.). A Study of the Generalization Capabilities of XCS, 8.

Lanzi, P. L., & Loiacono, D. (2006). Standard and averaging reinforcement learning in XCS, 1489.

Laplace. (1818). *Deuxieme Supplement a la Theorie Analytique des Probabthtes (Courtier, Paris); reprinted (1847) in Oeuures Completes de Laplace,* (Vol. 7).

M. L. Puterman. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley.

Maia, T. V. (2009). Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective, & Behavioral Neuroscience*, *9*(4), 343–364.

Nakada, T., & Takadama, K. (2013). Analysis of the number of XCS agents in agent-based computational finance, 8–13.

R. J. Urbanowicz and J. H. Moore. (2009). “Learning Classifier Systems: A Complete Introduction, Review, and Roadmap,” Journal of Artificial Evolution and Applications, vol. 2009, pp. 1-25.

Richard S. Sutton, A. G. B. (2017). *Reinforcement Learning: An Introduction*. The MIT Press Cambridge, Massachusetts London, England.

S. Haykin. (1999). *NEURAL NETWORKS, A Comprehensive Foundation, Second Edition*. PEARSON.

S. W. Wilson. (1986). “KNOWLEDGE GROWTH IN AN ARTIFICIAL ANIMAL,” presented at the Proceedings of the 1st International Conference on Genetic Algorithms.

S. W. Wilson. (1991). “The Animat Path to AI” From Animals to Animats: Proceedings of the First International Conference on the Simulation of Adaptive Behavior, Cambridge, Massachusetts: The MIT Press/Bradford Books.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., … Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Stewart W, W. (2007). ZCS: A Zeroth Level Classifier System | Evolutionary Computation | MIT Press Journals. Retrieved April 3, 2018.

Tesfatsion, L. (2000). Introduction to the JEDC Special Issue on Agent-Based Computational Economics, 17.

Thompson, J. Mark. (2000). “Defining the Abstract.” The Games Journal.

Urbanowicz, R. J., & Moore, J. H. (2009). Learning Classifier Systems: A Complete Introduction, Review, and Roadmap. *Journal of Artificial Evolution and Applications*, *2009*, 1–25.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning*, *2*, 199–228.

Wilson, S. W. (1994). ZCS: A Zeroth Level Classifier System. *Evolutionary Computation*, *2*(1), 1–18.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (1999). State of XCS Classifier System Research, 22.

Wilson, S. W., Wilson, S. W., Xcs, G., & System, C. (1998). *Generalization in the XCS Classifier System*.

Zang, Z., Li, D., Wang, J., & Xia, D. (2013). Learning classifier system with average reward reinforcement learning. *Knowledge-Based Systems*, *40*(Supplement C), 58–71.

Zenobia, B., Weber, C., & Daim, T. (2009). Artificial markets: A review and assessment of a new venue for innovation research. *Technovation*, *29*, 338–350.

APPENDIX A

The following code snippet shows the main loop that specifies many sub-procedures for learning in XCS. The definition of parameters and code initialization is done beforehand. For an in-depth-description, please confer to (Butz & Wilson, 2002).

```
while ENV(agent_location(1),agent_location(2))~='F'
    actual_time=actual_time+1;
    [XCS_dv] = GetSituation(ENV,agent_location);                    %get situation (env); In this part, agent location in environment is determined.
    [match_set,classifiers_population] = GenerateMatchSet(classifiers_population,XCS_dv,theta_mna,p_dont_care, p_I,epsilon_I,F_I,...
                                                           actual_time,N,theta_del,delta);    %generate match set [M]
    [prediction_array]=GenereatePredictionArray(match_set);         %generate prediction array PA out of [M]
    [act,exploit_parameter] = SelectAction(prediction_array,p_exp);  %select action according to PA
    [action_set] = GenerateActionSet(match_set,act);
    [agent_location,location_time_step,steps_to_food,Ro] = ExecuteAction(ENV,act,agent_location,location_time_step,exploit_parameter,...
                                                             steps_to_food); %Action execution and getting reward

    if size(previous_action_set,2)~=0
        betaRO = betaRO -((betaRo_max - betaRo_min)/number_of_problems);

        RO_Rlearning = RO_Rlearning + betaRO*(previous_reward +...
            max(prediction_array)- max(previous_prediction_array) - RO_Rlearning);  %Required parameters for calculating P in R-learning

        P_XCSAR = previous_reward - RO_Rlearning + max(prediction_array);    %P value for both single method is calculated
        P_XCSG = previous_reward + discount*max(prediction_array);
        P =(P_XCSAR + P_XCSG)/2;                                             %Here P value is calculated based on average of two single methods
        P_XCSAR_vect(index)=P_XCSAR;
        P_XCSG_vect(index)=P_XCSG;
        P_vect(index)=(P_XCSAR + P_XCSG)/2;
        index=index+1;
        [previous_action_set,classifiers_population] = UpdateSet(previous_action_set,P,beta,epsilon_zero,alpha,nu,classifiers_population,doActionSetSubsumption,...
                                                                  theta_sub,doGradient); % Update function for updating classifier parameters
        [classifiers_population] = RunGA(previous_action_set,actual_time,classifiers_population,theta_GA,kappa,doGASubSumption,previous_XCS_dv,...
                                                                 mu,N,theta_del,delta,theta_sub,epsilon_zero,N_sp,P_sp,dospecify); %The genetic algo. in XCS
    end
```

The main function of the XCS genetic algorithm (Butz & Wilson, 2002).

```
function [classifiers_population] = RunGA(action_set,actual_time,classifiers_population,theta_GA,kappa,doGASubSumption,XCS_dv,mu,N,theta_del,...
                                          delta,theta_sub,epsilon_zero,N_sp,P_sp,dospecify)
 sum_tsXnumcl=0;
 sum_numcl=0;
 for i=1:size(action_set,2)
   sum_tsXnumcl=sum_tsXnumcl+action_set{i}{7}*action_set{i}{9};
   sum_numcl=sum_numcl+action_set{i}{9};
 end

 if actual_time - sum_tsXnumcl/ sum_numcl > theta_GA
    for i=1:size(action_set,2)
      action_set{i}{7}= actual_time;
    end
    [parent1] = SelectOffSpring(action_set);
    [parent2] = SelectOffSpring(action_set);
    child1=parent1;
    child2=parent2;
    child1{9}=1;
    child2{9}=1;
    child1{6}=0;
    child2{6}=0;
    if (rand<kappa)
        [child1,child2] = ApplyCrossOver(child1,child2);
        child1{3}=(parent1{3}+parent2{3})/2;
        child1{4}=(parent1{4}+parent2{4})/2;
        child1{5}=(parent1{5}+parent2{5})/2;
        child2{3}=child1{3};
        child2{4}=child1{4};
        child2{5}=child1{5};
    end
```

```
child1{5}=child1{5}*0.1;
child2{5}=child2{5}*0.1;
%Child1
if dospecify==1
[classifiers_population] = specify_func(action_set,N_sp,P_sp,XCS_dv,actual_time,classifiers_population,N,theta_del,delta);
end
[child1] = ApplyMutation(child1,XCS_dv,mu);
if doGASubSumption==1
    [DoesSubValue1] = DoesSubsume(parent1,child1,theta_sub,epsilon_zero);
    [DoesSubValue2] = DoesSubsume(parent2,child1,theta_sub,epsilon_zero);
    if DoesSubValue1==1
      parent1{9}=parent1{9}+1;
    elseif DoesSubValue2==1
      parent2{9}=parent2{9}+1;
    else
      [classifiers_population] = InsertInPopulation(child1,classifiers_population);
    end
else
    [classifiers_population] = InsertInPopulation(child1,classifiers_population);
end
[classifiers_population] = DeleteFromPopulation(classifiers_population,N,theta_del,delta);
```